MASTER IN COMPUTER SCIENCE

Implementation of belief revision on Sony Aibo

Vander Schelden, Yves

Award date:
2004

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX, NAMUR
Institut d'Informatique
Promotion 2004

Implementation of Belief Revision on Sony Aibo

Yves vander Schelden

Graduation Thesis for Master Degree

Abstract

This paper proposes an implementation for a system of Belief Revision for the robot Aibo of Sony.

Belief Revision is an impressive theory of nonmonotonic reasoning based on the AGM paradigm, which could lead to amazing progress in fields of artificial intelligence such as behaviour simulation. This paper retraces the important bases of simple logic, nonmonotonic reasoning and belief revision. It describes one way to implement such a system for Aibo, using two major Australian belief revision systems.

Some belief revision applications are built and discussed in different application domains.

Résumé

Ce mémoire propose une implémentation pour un système de "Belief Revision" (révision de croyances) pour le robot Aibo de Sony.

L'impressionnante théorie des "Belief Revision", basée sur le paradigme AGM, pourrait nous mener à d'incroyable progrès dans le domaine de l'intelligence artificielle comme les simulations de comportements. Cet écrit retrace les bases nécessaires de logiques simples, de logiques non-monotones et la theorie des "Belief Revision". Il décrit une manière d'implémenter un tel système pour Aibo, en utilisant deux importants systèmes de "Belief Revision" australiens.

Nous mettons en en oeuvres et discutons d'applications dans différents domaines.

This Master Degree work demanded a fair amount of personal investment, but it couldn't have been achieved without the help of several people. That's why, I would first of all like to thank my Professor, Mr Schobbens, for his support and guidance during the whole process of my work.
I would also like to thank my supervisor in Australia, Pr Mary-Anne Williams, for all her help during my stay in Australia. Although she was very busy, she managed to find some time to orientate and follow my research.
My gratitude also goes the whole team of UTS Unleashed for their welcome, kindness and support with the Aibo implementation.
I would also like to thank my rereaders Ian Sollars, Christine Moeys and Julien Moeys, who helped me to correct a huge number of English mistakes.
Besides, I would like to say a special thanks to Frederic Randolet, who supported me and helped me for the redaction.
And last but not least, I would especially like to thank all my family and my friends for their unconditional support.

Table des matières

Introduction

A lot of people wish for a beautiful life where work could only be an occupation, not compulsory labour, where they could just enjoy life, doing only what they want to do. We all know that some work is off-putting, other is very interesting, like my own. In present, with our computer science achievements, we should be able to delegate a lot of this less interesting work to a computer. In fact, the world of computing is expanding faster than any other environment. Why don't we already have some kind of artificial intelligence (AI), passing the Turing Test[1], to interact with us for the worst repetitive work ? Why haven't we already succeed in this direction ?

A response can be drawn from the fact that the task of coping with this problem is very heavy because the human brain is the most complex structure we know and because the scope is too large. Replicating this complex organization is rather impossible, or at least, it would be the most difficult work to do and to accept as a reduction of our own humanity... but our goal is not in here. From the engineering perspective of the previous questions, defining some kind of environment where we could limit our research is the first thing to do.

Having some kind of playground with a specific guinea-pig could boost our comprehension. We then found a perfect place in the University Technology Sydney(UTS) magic lab, with the Sony Aibo dog to play with. This little robot, provided with a powerful processor and a lot of sensors, is a good example. The restricted world of soccer competition and playground is a very convenient environment to start with. That is what we have done in Australia.

Human behaviour has different ways of achieving a goal : reasoning, wisdom, innate reflexes and so on. First of all, it could be possible to simulate our reasoning by deduction. The other ways could be approached in another

[1]Weizenbaum illustrates it in [WEI76], other information available on [TUR36]

paradigm. For deduction, our goal can be achieved by simple logic, as has already been studied for years computer science, completely based on it. Thus, we need to remember some of these principles before presenting our work with Belief Revision.

One of the most impressive achievements is the implementation of Theorem Provers, the way we implement the reasoning and then deduce all propositions from a specific theory base, by simulated computation. These Theorem Provers are the key for a mathematical and logical derivation methodology done by a computer. Thus, this will be used for our logic development.

Simple logic is efficient but not sufficient to achieve our goal as we need a field with larger possibilities that we met with nonmonotonic reasoning. This extension is used to let us work with exceptions, with rules that can be overridden in particular cases. Belief Revision is a specific system of the nonmonotonic reasoning theories that we chose for this implementation. We shall see the differences with the other main systems. In other words, we have different ways of resolving these exceptions and Belief Revision derivation should be, in our particular viewpoint, a specifically good resolution with a human-like reasoning. To this end, we tried to link a belief revision(BR) system as a plug-in for Aibo, providing new faculties (may we say intelligence) to this little puppy.

Even though apparently simple, the system we developed is rather complex. We tried to have native code running on Aibo's operating system. But for implementation reasons, we limited our application on a de-localised server to process intelligence. A specific protocol has been implemented and it is possible to access a particular representation of Aibo's mind at any time. Other use of the developed applications are also expected such as splitting the reasoning process between some agents.

I also give some academic examples that illustrates the interest of our system. In the future future, these dogs could have some kind of computerized way of belief, a behaviour that all psychologists could dream of, fully analyzable and with no cheating or other strange human phenomena.

We'll speak about all these subjects in the following chapter.

Chapitre 1

State of the Art

In this chapter, we will first discuss about the context in which we will work and then coverer this domain's state of the art.

1.1 Aibo and Co.

A possible application of usual artificial intelligence techniques is conducted with robots. In my research framework, I've looked specifically at the Aibo[1] puppy problem, with a Belief Revision solution. Let's start discussing the development already done on the subject and about the foundation of this idea of intelligence, which incorporates a set of different concepts.

FIG. 1.1 – Aibo - used version for this report.

[1] http ://www.sony.net/Products/aibo/

1.1.1 Scope

Based at the UTS university, the UTS Unleashed Team developed a full-service intelligence system required for the Aibo puppy (Figure 1.1) to succeed in playing a soccer match autonomously. It implies coordinating its physical (engine) and also logical (strategy and tactics) skills, taking into account his teammates positions and states. This initiative is being pursued by different universities around the world, competing together in a soccer competition called RoboCup[1](See Figure 1.2). The main challenge of this development is to interpret the set of stimuli coming from the outside, convert it to a pseudo-mental scheme, compute a plausible behaviour for the soccer player - following a given strategy - and to coordinate the required movements (engine running and feedback from the outer world) in order to respond to a specific situation. Thus, different sub-projects can be perceived in this work.

FIG. 1.2 – Aibo - An aibo playground.

First, we need to convert raw images from one or more cameras and link it with different objects, by shape, colour and size recognition. The process also includes the distance computation which is harder to achieve since we should have only one camera (like the Aibo we worked with). Situations where Aibo doesn't see the whole, partly hidden by an other object, are also complex.

[1]http ://www.robocup.org/

A second part consists in translating the position of all this visual information into a mental schema or world representation. We could have other sources of information than vision like sound or touch via the sensors. These other sources could be studied by another team.

Then there is an engineering part, concerning the coordination of different engines of the robot, corresponding to movements sufficiently fast and quite safe[1] in general. The simple fact that the robot's engine is running while the vision didn't record any displacement is a very important fact, as it tells us that there is an object in the way that stuck the Aibo and is out of its field of vision. This is the case when Aibo reaches a wall that it can't discern or reaches some kind of border. It seems like a basic case, but this is not a conclusion that reached by most A.I. systems implemented today. Once all these physical considerations are managed, we define all the actions for the robot to do, in order to achieve some specific goal (such as winning a football game) with convincing behaviour. The considerable work that UTS Unleashed does is constantly improved. Different papers can be read to get more details as for example the two last reports : [UUR02] and [UUR03].

1.1.2 The Idea

This reasoning constitutes a first step in Artificial Intelligence and provides a first understanding towards an application of the mechanism of this domain ; the field of Informatics. Indeed, beginning with a concrete example of a specific domain, gives us a first impression of the general complexity of the problem we face. Moreover, it can help us to build a systematic walkthrough, also define a method to control all possible errors, since we develop the system step by step. In the worst case, if the method and structure we choose is not satisfactory, we will learn from our errors and start again with a better experience and a more adaptive approach to developing and a system of intelligence. In general, we characterize the intelligence scheme that could be generalized, as we already do in the mathematics method of deduction to demonstrate a theorem.

[1] safe : the robot must avoid obstacle and any crash.

1.1.3 The Target

Our starting point will be the creation of an intelligence system that can deal with a soccer players needs. Keeping in mind that we want a robot that can work in all situations and interact with a non-specific world, it is probably a good idea to start with a practical application such as the restrained, relatively simple world of soccer. The game of soccer requires some obvious objects like a goal, a ball, a field limit, and some puppies with opponents and team mates, constituting a kind of sandpit for our little pet.

1.1.4 Actual AI Development

The actual development is based, as most of artificial intelligence is up to now, on pre-compiled code without any learning ability[1]. Two different approaches are usually adopted for this.
The first aims at describing any situation by its own characteristics. The formalism used for this description can be designed with more or less arguments, to give more or less detail. Surely, the more arguments we have, the more precise the "description of the situation" is. Once done, we can link any situation to an expected response. Aside from just being a database of pre-compiled cases, this technique has one big problem as follows : To improve intelligence, one increases the definition of each state. But for each argument added, we at best double the number of cases. We then have an exponentially growing set to link with a predicted behaviour.
The second way to define a good behaviour is to specify a few rules which shows the best thing to do for a given feature. Each rule is coupled with a value bringing the importance, the probability for this prediction to be the most interesting step to do.
Other approaches to pre-compiled implementations that we don't define here. Anyway, the problem with these solutions is that we always need to rebuild the code for any change we want and we need to choose the basic reasoning of the machine once, for all cases. Some kind of runtime-reconfiguration of reasoning appears to be necessary at this step. The community also wants a machine that could deduce the rules itself "from scratch" or from all information already provided.

[1] it is a simplification : developments such as Lisp are also notable.

1.2 Different Approaches

The parallel approach that we keep in mind is to bring more than just one specific behaviour of soccer players to the dog. We want to give him the ability to manage various situations without any help. Our dream is to achieve good behaviour of the real world ! To do so, we had to choose between some different possibilities, none especially exclusive. We discuss here these different methods and their inconveniences.

1.2.1 The Semantic Web

The first approach was to use the semantic web that we describe in the paragraph below. We first consider all entities we work with as generic objects. After a comparison with a huge object database that constitutes a main feature of the semantic web, we link a given world object with its logical representation. Once the information is provided by the system, we have access to useful knowledge such as the description of its interest, physical characteristics and main uses[1]. [further explanations of sem. web] This approach seems really promising, but we don't use this technology because the actual technology isn't yet developed and standardized enough for the goal we want to achieve. This science is still in its infancy. Progress in this field is made and reported in [SemWeb] and [W3CSW].

1.2.2 The Physics Engine

Other tactics consists of defining the principal physical concepts and all logic postulates that these imply. To illustrate this idea, let's look at our main example : soccer. The idea of a rolling ball, of a sliding ball, or the idea of a ball blocked by an obstacle is for us a tautology. We know this by our own experience as we saw all this during our childhood. Surely, for Aibo, a synthetic being, these experimental notions are not as easy as they look. The human brain is made in order to accumulate any kind of experience that brings a specific empirical reasoning as it grows. That sort of behaviour is only possible with a learning system such as our brain. Indeed, only a learning process such as a neural network (or similar applications) should be able to perform these deductions, at least in actual development.

[1] described in programmed methods

Furthermore, a lot of study has been done in this direction. This vision of intelligent improvement then consists of furnishing concepts like gravity, the action-reaction principle and so on. How can we define what's going on when we push a ball ? It is an obvious fact for us that if we push a block[1] from one place to another, we will use some energy, due to ground friction (which is due to gravity[2]). It's also obvious that an object won't go through the table because of the action-reaction principle (and strong electromechanical force between atoms). These are the physical concepts we need to implement. We have to provide our robot with the possibility of using this information. We don't need to take these hard examples into account. The following simpler example is sufficient to have an idea of the complexity of this state.

In the soccer world, we need to know where a ball goeswhen we kick it. For any non-intelligent system, kicking the ball is just kicking the ball. All the consequences are unknown. It would be more interesting if we knew where the ball is going (direction, distance, heightĚ) in this case, which depends on kick power and velocity. The creation of these rules is a difficult task as our own reasoning is conducted by empiricism. Flux is the system that we were interested in, which is an inference engine that can be used for a physical approach to concepts like the well-known Havoc system for video games[3]. Flux has been coded in Prolog by the university of Dresden, Germany[Flux]. Once again, we dismiss this work direction for the moment to focus on a more practical yet abstract system called nonmonotonic reasoning.

1.2.3 Belief Revision

In fact, our main idea is to create a method that provides real-time behaviour computation. The robot has to base its reflection on its own. As it has got some sense, he will take some info from the outside world. The theory

[1] we don't matter what kind of block it is in this level. It could be concrete or wood, big or small ...

[2] we don't need to have this perfection of knowledge :
a child just knows that this could be difficult to push the block.
That's all what we want in this level.

[3] www.havoc.com

base will then contain all facts and rules received before and the information given from external help or being derived within the logic system. In the present time, the system should make two type of reasoning :

- Create logical rules from the current axioms in the database.
- Associate sensations to facts in a theory base.

We can imagine having empirical reasoning later, on the basis of sensed information.

To be more concrete, we give some information : rules and facts. The system can also draw other facts from the real world. From this set of information, it is able to derive all the propositions it can make with any combination of reasoning. This theory base can then evolve : by adding new information, removing previously deducted errors, and confronting different opposing theories. In order to build such a system, Belief Revision(BR) is a good method. It permits us to attribute a degree of information certainty to any rules and facts.

Let's take this little scenario to fix our idea : A student lives with his father who is woodcutter and with his mother, a university professor. As he is worried about his exams, he asks his parents for some advice. The father says that he has to study until early in the morning to succeed, even if he doesn't sleep at all, because he has to know the whole subject. This advice is fairly reliable although the man didn't get any higher education. The mother has another point of view. She thinks that the fitter the child is the better he will succeed. In other words, she would rather have him sleep well than studying too much before the exam. She thinks that the most important thing is managing to prove what he really knows. The woman knows more about university, so the wise guy would attach more credit to his Mom. Thus we can represent our knowledge base in the following schema :

$$
\begin{array}{ll}
Knowing\_everything \rightarrow Study\_all\_the\_night & certainty : 0.5 \\
Succeed \rightarrow Knowing\_everything & certainty : 0.7 \\
Ability\_to\_restitute \rightarrow Sleep\_well\_and\_enough & certainty : 0.9 \\
Succeed \rightarrow Ability\_to\_Recall & certainty : 0.7
\end{array}
$$

Moreover, the student surely knows that he needs to choose one of these 2 exclusive solutions. In fact, mixing these four rules leads to what we call inconsistency. This consistency problem often occurs in great systems due to all "crossed" interactions and that is why we need such resolution. In our example, the student will choose to sleep instead of staying up to study all night, a logical choice indeed. We shall see during the following chapter how he can choose without any external help by nonmonotonic reasoning. In fact,

we can see that a part of the knowledge base will be obsolete and dismissed. Indeed, belief revision will contract the theory base at each step and keep only the needed information, not the deduction. The contradiction will also be removed. In short, we obtain a progressive system that enables us to mime some processes of reasoning, moving step by step to a better solution. It stands to reason that the application of Belief Revision in a serious environment will lead to a very complex architecture, and a head-ache. In order to develop consistent systems, we need some practice and some academic examples.

1.3 Major AI Interest

Artificial intelligence has been developed for many years for different reasons. We will here underline a few ideas that require more development. We wanted also to give you some ideas how industry could invest in this area of scientific improvement.
This kind of progress is made by *learning systems*, *data analysis and manipulation*, *pattern recognition*, etc.

Impressive work has been done in the development of a major system of artificial intelligence called directIA[DirectIA] by the French team MASA[MASA]. This system is able to work for many purpose as military, entertainment, modelling systems, ...

Entertainment

The video game industry conducts a lot of surveys in the domain. In fact, we use artificial intelligence to mime a human player when you're playing alone. That was an essential need before the Internet arrived at home but nowadays it is still important for a game. It includes strategy and tactics, human-like failures, "intuition" (or decision forecast) and other skills.

Expert Systems

Expert systems are used to help any human to choose the best solution for a given problem. For example, a CEO needs to choose to develop one product or another, the system can take all information about world offer and demand, about production needs like resources cost and supply, about logistic and so on. It will then give some advices regarding different given decision rules.

Expert systems are useful for decision processes[1]. The major techniques used for this computation are matrix calculus and learning machine.

Military

Military domain is a great investor in AI. The development targets autonomous control, like missile guidance and target identification for instance.

Robotics and Route Planning

It is possible to use AI to organize every movement of a robot.. Such systems may also include a decision system, to help a robot to get out from any critical situation. The most important thing is to give the robot an autonomous belief system.
Route planning is the way to compute how we can go from a place to another, regarding any obstacle in way and with some conditions. The GPS[2] system is a good example of it.

Modelling System

System modelling is an important tool to represent any specific domain. A major use for it is to predict how a system will evolve after a given action. It is often used in domain like chemistry, biology or astronomy.

[1] logistic, administration or factory improvement for instance
[2] The Global Positioning System

Chapitre 2

Theories

Before talking about the concrete application, the main part of this thesis, we need to remind the reader about different theories in order to understand all subtleties of this method. We shall start from the simplest and go to the hardest theories : first a short reminder of basic logic, then we will take a look at nonmonotonic reasoning, and finally to the Belief Revision Theory.

2.1 Basic Logic[1]

First, let's start with Basic Logic. A basic logic is a representation of the world and of the knowledge that permits us to deduce and prove some propositions in a systematic and deterministic manner. This representation is based on formulae, rules and facts, assuming that we also have a deductive algorithmic principle. It allows us to model some sensible behaviour. Different kinds of logic exist for example : propositional logic, first-order logic, second-order logic and higher-order logic. These are more and more powerful generalizations. To define a logic, we need to define a syntax in a formal way then we need to link it with semantics, also called declarative semantics. Then, we need to assign an algorithmic walkthrough (rules) to prove any other formulae, also called procedural algorithmics, that is useful to deduce other more interesting rules from simplest ones.

[1]This information came from my courses[JMJ-C] and a website[MAT-W]

For further development, it could be useful to go through the informal definition of the concepts of interpretations and models.

Interpretation : set of simple propositions (true propositions)

An **interpretation** I is a model of a proposition p
iif the truth value of p regarding I is true.

An **interpretation** I is a model of a set of propositions
iif I is an interpretation of each propositions of the set.

2.1.1 Classic Proposition Logic (or boolean calculus)

Abstract Syntax[1]

let $p, q \in \mathbb{P}$ (Atomic predicates)
$< bin >::= \wedge | \vee | \Rightarrow | \leftarrow | \Leftrightarrow | \ldots$
$< un >::= \neg | \ldots$
$< formulae >::= p | < \Phi_1 >< bin >< \Phi_2 > | < un >< \Phi >$
where Φ,Φ_1 and Φ_2 are Formulae[2]

This is not an unique representation. We can complete the "..." by other new symbols and link them to their own semantic. In fact, we just need two atomic symbols, the *constructors* : $\wedge$ and $\neg$. The others are unnecessary but their manipulation seems easier for the beginner.

Indeed, we can derive the other symbols from the two constructors :

$$\Phi_1 \vee \Phi_2 \equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2) \quad | \quad \Phi_1 \Rightarrow \Phi_2 \equiv \neg\Phi_1 \vee \Phi_2$$

To remove ambiguity, we still need a priority within the different operations :

$$\neg, \wedge, \vee \leftarrow, \Rightarrow, \Leftrightarrow$$

Priority has the same interpretation as arithmetic priority in algebra. We compute the negation first for example in $\neg a \vee b$. Using the same idea, some operators are attached to their variables and computed in combination with another[3].

[1] abstract syntax doesn't define some of the simplest characteristics of a language as bracket impacts.

[2] considering brackets as already defined

[3] like the law of distributivity in mathematics

Semantic

formulae interpretation ≡ application from atomic predicate to $\{true, false\}$
ie. $I : \mathbb{P} \longrightarrow \mathbb{B}$
We have always[1] :
$I \models \emptyset$
$I \models p \quad iif \quad I(p) = true$
$I \models \Phi_1 \wedge \Phi_2 \quad iif \quad (I \models \Phi_1) \wedge (I \models \Phi_2)$
$I \models \neg\Phi_1 \quad iif \quad I \not\models \Phi_1$
Φ is valid $\stackrel{\Delta}{\equiv} \forall I, I \models \Phi \quad \equiv \quad \models \Phi$
Φ is satisfiable $\stackrel{\Delta}{\equiv} \exists I, I \models \Phi$

Proof Rules

These are some of the most usual[2] :

Axioms :	Meta-Rules	
$p \Rightarrow (q \Rightarrow p)$	$\frac{p \quad \wedge \quad p \Rightarrow q}{q}$	(Modus ponens)
$(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$	$\frac{\neg q \quad \wedge \quad p \Rightarrow q}{\neg p}$	(Modus tollens)
$(\neg q \Rightarrow \neg p) \Rightarrow (p \Rightarrow q)$	$\frac{p \Rightarrow q \quad \wedge \quad q \Rightarrow r}{p \Rightarrow r}$	(Chaining)

Regarding these different meta-rules, we can prove that the number of deductable proofs are countable. We can for instance take the simplest algorithm (semi-algorithm in fact) in enumerating all proof possibilities. This algorithm is computable and NP-complete.

The other well-known formula could be useful to quicken the process but are derivable from the three meta-rules above. You can access it in [MAT-W][propositional calculus].

---

[1] note that $\models$ is a symbol for declarative satisfaction or equivalence.
In fact, a proposition F is a logic consequence of a set of proposition S iif all model of S is a model of F $\equiv S \models F$

[2] for meta rules, numerator is antecedent and denominator is consequent

2.1.2 First-Order Logic[1]

The purpose of this section is to present important differences with other formalisms. These description are not exhaustive.

Abstract Syntax

let $p, q \in \mathcal{P}$
let $f \in \mathcal{F}$
$\Phi ::= p(t_1, \ldots, t_n) | \Phi_1 \wedge \Phi_2 | \neg\Phi_1 | \forall x.\Phi_1$
where terms are $t ::= x | f(t_1, \ldots, t_n)$

Note that the formula Φ is the simplest one, but we could also add the well-known $\exists$ symbols such that $\exists x.\Phi_1 \triangleq \neg\forall\neg\Phi_1$[2]

Semantic

$$Interpretation : \left\{ \begin{array}{ll} \mathcal{V} \rightarrow \mathcal{D} & \text{(Values)} \\ \mathcal{D} \neq \emptyset & \text{(Domain)} \\ \left. \begin{array}{l} \mathcal{P} \leftarrow (\mathcal{D}^n \rightarrow \mathbb{B}) \\ \mathcal{P} \leftarrow (\mathcal{D}^n \rightarrow \mathcal{D}) \end{array} \right\} & (n = \alpha(p)) \end{array} \right.$$

$I \models \emptyset$
$I \models p(\bar{t})$[3] iif $I(p)(I(\bar{t})) = true$
$I_{\mathcal{V}}(x) = I_{\mathcal{D}}(x)$
$I(f(\bar{t})) < I(f)(I(\bar{t}))$
$I \models \Phi_1 \wedge \Phi_2$ iif $(I \models \Phi_1) \wedge (I \models \Phi_2)$
$I \models \neg\Phi_1$ iif $I \not\models \Phi_1$
$I \models \forall x.\Phi \triangleq$ for all $d \in \mathcal{D}.I[x \leftarrow d] \models \Phi$

---

[1] $\mathcal{P}$ are first-order predicates : $p(t_1, t_2, ..., t_n) : p \rightarrow \mathbb{N}$
$\mathcal{F}$ are functions
Φ and Φ_1 are formulae (recursive definition)

[2] say for example the phrase : "all chess pieces in game are white".
It's equivalent to "there is not black chess pieces anymore in the game".

[3] $\bar{t} \triangleq (t_1, \ldots, t_n)$

Proof Rules

The previous rules from Classical propositional logic can be extended by these two meta-rules :

$$\frac{p \Rightarrow q(x)}{p \Rightarrow \forall x q(x)} \qquad \frac{p(x) \Rightarrow q}{\exists x p(x) \Rightarrow q}$$

2.1.3 Transformation to Second-Order Logic

Abstract Syntax

Abstract syntax is nearly equivalent to first-order logic, with propositions that can have predicates as arguments. In other words, $\Phi ::= ... | \forall p \in \mathcal{P}.\Phi_1$

Semantic

It implies some new semantic :
$I \models \forall p.\emptyset$
$I[x \leftarrow d] \models \emptyset$
$I[P \leftarrow a] \models \emptyset$ for all $a \in (\mathcal{D}^n \rightarrow \mathbb{B})$

Proof rules

The one interesting added rule is : $\frac{\forall p.\Phi}{\Phi[p(x_1,...,x_n) \leftarrow \Psi(x_1,...,x_n)]}$

Example : Let $\Phi = \forall x.q(y)$
we can change it to : $q(x) \leftarrow \forall y.r(y)$
$\Rightarrow \forall x.\forall y.r(y)$

2.1.4 Transformation to Higher-Order Logic

Abstract Syntax

The only thing we need to keep in mind is that it gives in our system the possibility of recursive and overlapping functions.
ie. $t ::= x | f(t_1, \ldots, t_n)$

2.1.5 Different Kinds of Semantics

Now a word on the two different kinds of semantic used above. The first one is declarative semantic($\models$) and the second one is procedural semantic($\vdash$).

For terminology, we say that a proof system is valid if $\vdash \Phi \Rightarrow \models \Phi$.
The system is complete if $\models \Phi \Rightarrow \vdash \Phi$.

To have a more pragmatic vision, the declarative way is the method by which you can set a truth table for any expression. For the human brain, it's more understandable to have an approach of testing all possibilities for two expressions and check if the result is the same.

Procedural semantics are used to give exact reasoning to prove some expression. It's a harder way for human brain, but computers do it easily. That's why we need the two processes.

2.1.6 Example

Let just take a look to the famous bird sample. We know that a bird can fly. How can we deduce that an object is a bird ?

$$\forall x. fly(x) \Rightarrow bird(x)$$

Let $fly(Leon)$, or in natural language : Leon can fly.
The system will instantiate the variable x such that $x = Leon$ and that $fly(x)$
It follows that $bird(Leon)$, or in natural language : Leon is a bird.

The problem we'll need to face in the following chapter is : How can we accept that Mike could be a penguin, bearing in mind that penguins can't fly, or in logic :

$$x = Mike \wedge bird(x) \wedge \neg fly(x)$$

2.2 Nonmonotonic Reasoning[1]

After different trials it has been shown that simple logic is not enough to simulate human reasoning.
Here is an informal definition[2] of non-monotonic logic :

When declaiming a line of reasoning, it is the custom to present a case for the object of discourse. During the argument, the propositions are steadily built up, monotonically increasing. But, in non-monotonic logic, the propositions are defeasible; that is, if an impasse is reached, the propositions can be abandoned; thus the number of valid propositions no longer has to increase steadily, but can even decrease and further fluctuate.
This form of reasoning can be thus used to model thought, as in the scientific method, where hypothetical explanations can be abandoned in the light of further evidence from observation, inference and experiment.

A textbook case can be mentioned such as : the frame problem, the Yale shooting, the ramification problem, the qualification problem or the default priority. Below are some explanations of these problems.

2.2.1 Classic Examples

Frame Problem[3]

One main practice in Artificial Intelligence is the frame problem. The major question is "How can we describe all of the environment variations due to a specific action ?". This is a little sample to illustrate the problem.

A robot's reality is based on it's beliefs. They are divided in two groups : thinking and observation. The first is an internal process and the second is external information generated by sensors. The robot needs to represent this reality, that depends on the current time. The simplest way to manage this information is to use the notion of situations. All modifications of the robot world model can be described by actions. These actions can be done by himself or for another reason.

---

[1] This course [PYS] has been used all along this chapter
[2] comes from the wikipedia's article : www.thefreedictionnary.com[10/08/04/05 :31]
[3] Reference are : [HAY73] and [LIF94]

We assume that we can represent any situation from the starting situation s_0, applying a predicate on it each time an action occurs. For example, we will represent the action of building a brick tower for a child defined as follows : "You can put the brick on top of this one onto some other one, if that one has not got something else on it". That will give :

$$(on(b_1, b_2, s)\&\forall z.\neg on(z, b_3, s)) \supset on(b_1, b_3, R(move(b_2, b_3), s)).$$

To understand this, we need an explanation of the predicates and functions used :
"*on*" predicate says if the first argument is on the second argument in the given situation. "*move*" is the action for moving the first block on the second block. "R is" the application of an action to a given situation. "$\supset$" is the implication symbol.
We could also describe the situation to move a robot in a certain direction. Starting from a situation s_0, moving the robot forward is simply described as $R(moveforward(distanced), s_0)$.

The problems occurs here : we know all the changes of the situation, but what about the other objects ? We don't know anything about the other bricks. We don't know anything about the robot world model. We don't know if an earthquake happens or anything else. We don't even know if the top of the brick tower is still in place.

An obvious solution to the frame problem is to add special axioms to the domain representation, called frame axioms (or frame fluent) that explicitly list what is not changed by each action. In other word we can create a fluent, variable predicate or function describing any property (for example : colour(object o, situation s)=yellow) and which could be "more static" than a normal predicate. Clearly, in complex domains, the list of frame axioms quickly becomes intractably large. We could try to make a generic axiom setting all frames fluents to the default value. The problem is then the same as this axiom should change from time to time.

This approach of non-monotonic logic can break the problem because the rules are not always true, but true by default. A lot of other specific frames were encountered as described below.

Yale Shooting

Another part of the frame problem is the Yale shooting problem.
It concerns a person who at any point in time is either alive or dead, and a gun that can be either loaded or unloaded. The gun becomes loaded any time a "load" action is executed. The person becomes dead any time he is shot with a loaded gun.
Assume that the person is initially alive. The gun is loaded, then he waits for a while, and then he is shot with the gun. What can we say, given these assumptions, about the values of the fluents involved – "alive" and "loaded" – at various points in time?
The description of the domain above does not say whether the "load" action is considered executable when the gun is already loaded. Let's decide that the answer is yes : when the gun is loaded, that action can be executed but will have no effect.
Note that the assumptions of the Yale Shooting Problem do not determine the initial state completely : the fluent "loaded" can be either true or false in the initial state. But once the initial value of this fluent is selected, all future changes in the values of fluents are uniquely defined.

We still have the problem encountered in the previous section. But another problem arises as we wanted to minimize the number of of possible change changes. Let's take a look to these two example regarding our fluents and their possible exceptions :

1	$\begin{matrix} alive \\ \neg loaded \end{matrix}$	$+ load \rightarrow$	$\begin{matrix} alive \\ \underline{loaded} \end{matrix}$	$+ wait \rightarrow$	$\begin{matrix} alive \\ loaded \end{matrix}$	$+ shoot \rightarrow$	$\underline{\neg alive}$
2	$\begin{matrix} alive \\ \neg loaded \end{matrix}$	$+ load \rightarrow$	$\begin{matrix} alive \\ \underline{loaded} \end{matrix}$	$+ wait \rightarrow$	$\begin{matrix} alive \\ \underline{\neg loaded} \end{matrix}$	$+ shoot \rightarrow$	$alive$

Here, we'll assume that the gun can't be reloaded by external action[1]. In both sample we can imagine for some reason that the loading action did not occur for some reason. The system has an exception on both sides. Exceptions are the special facts that change the fluent "normal" behaviour. Let's continue the description. Suppose that the gun is loaded, we describe that nothing happens during the delay and we shoot the guy. An exception is possible when we shoot the guy and he miraculously survives. We have in this situation(1) an exception on the alive fluent. In situation (2), we decide that the exception

[1]the other possibilities will lead to a similar result

occurs while waiting to shoot, the gun unloads (fire in the sky)[1]. Surely we can't then kill the guy since the gun is unloaded.
This sample is a bit weird to explain, but it shows that it is impossible to find a perfect situation with a perfect description for an action. The better representation should be the one which leads to the least exceptions and this representation doesn't exist.

Ramification Problem

If we take the simple Yale Shooting problem, we can add a new predicate in the rule $dead(i, s) \rightarrow \neg respire(i, s)$

This *respire* implies a new exception on $s_2 : \neg respire$ which introduces a new exception in the first description of the Yale Problem. Thus the second one will be chosen as a more logical possibility. In any case, it introduces another problem : adding a new exception to the description may not lead to the creation of a model any better than previous models.

We could also talk about the ramification problem which is similar but these examples are sufficiently representative of the monotonic logic problem. All these examples can be partially or completely resolved by the use of non-monotonic reasoning. Before enumerating the features of Belief Revision, we suggest we review the two most interesting methods of nonmonotonic reasoning : default logic which takes a syntactic view, and circumscription which takes the semantic approach.

2.2.2 Default Logic

This is a overview of The theory, as explained in [DPO94]

Formalization

Depending on the same syntax as *modus ponens*(1), the general rules presentation (2) can be transformed to describe the default rules(3).

$$\boxed{(1)\ \frac{a \quad a \rightarrow b}{b}} \qquad \boxed{(2)\ \frac{A_1 \quad \ldots \quad A_n}{C}} \qquad \boxed{(3)\ \frac{A_1 \quad \ldots \quad A_n \quad : \quad B_1 \quad \ldots \quad B_m}{C}}\ .$$

[1] remember that we don't know anything of the rest of the situation than just our fluent, due to the frame problem.

The third representation mean that $\{A_i\}_{i\in[1..n]}$ are the *preconditions*(or *antecedents*) and $\{B_i\}_{i\in[1..n]}$ are the *justifications* and C is the *consequent*(or *conclusion*) of the default.

This rule can be used only if we have already proved the antecedents and can't prove $\bigwedge_{i\in[1..m]} B_i$, then we can derive the consequent.

The main idea is to provide a system with a set of facts and a set of defaults. From this point of view, we extract all possible derivations from the system. The problem of justifications need to be verified. In other words, in all derivation step, we need to confront the new conclusion we choose with all previously encountered justifications. In this process, the choice of first default used is essential. That's why we have some different extension.

To be more precise and formal, a theory is a couple $\delta =< D, F >$[1].

The different steps are $s_0, s_1, \ldots$. Then $\bigcup_{i=0}^{\infty} S_i = S$ with $S_0 = F$.

S is a final extension and can be computed by

$$S_{i+1} = S_i \bigcup \left\{ \omega(\overline{c}) : \frac{\alpha(\overline{c}):\beta_1(\overline{c}),\ldots,\beta_n(\overline{c})}{\omega(\overline{c})} \text{ is an instance of a default in D with } \begin{cases} \alpha(\overline{c}) \Leftarrow S_i \\ \beta(\overline{c}) \text{is consistent with} S\ \forall \in [1..m] \end{cases} \right\}$$

Explanations

We can have more than one extension as we explain in this small example :

$$\text{Let F} = \{\text{P}\} \text{ and D} = \left\{\frac{P:\neg Q}{R}, \frac{P:\neg R}{Q}\right\}$$

The two possible extensions are :

$$\{\text{P,R}\} \text{ and } \{\text{P,Q}\}.$$

The first extension comes from the application of the first default on the factual set. After applying this default, we can't use the second assuming we need to preserve consistency.
The second extension is the converse. We apply first the second default rule and we are then stopped.

□

[1]where D are default, i.e. defeasible rules and F are facts.

These rules are the bases of default logic. Each extension is a model of the reality described in the theory base δ. It can be understood as an explanation of a formula.

Precisely, if g is a closed formula, the extension E is an explanation of g from $\delta =< D, F >$ if E is the set of consequents of some D'($D' \subset D$) when

1. $E \bigcup F \models g$;
2. $E \bigcup F$ entails the preconditions of D', such that we can order the default in D' and only use previous defaults in the sequence to prove the precondition of any default in D' ;
3. all of the justifications of D' are consistent with some extension of δ that contains E.

Some specific default profiles have been grouped to be computed faster and to predict easily how the system will behave as *normal* and *semi-normal defaults*[1] or *closed* and *open defaults*[2]. These different default types have characteristics which could be useful to read before actually using default logic.
For further information, please refer to [DPO94].

As we'll see in further examples, the way we choose to write our rules is absolutely critical. The extensions created will be totally different for each chosen syntax. Indeed, we need skill to obtain the foreseen result.

At last, to compute this research of a final extension, two main approaches are possible : the forward-chaining and the backward-chaining default prover. The first is an approach of generate and test, trying all combinations of default and getting a new extension when the system is blocked. The second approach is the converse, trying for a final formula to derive its predecessor until we have a way in the solution tree to explain this final point.

[1] normal default are of the form $\frac{\alpha(\overline{x}):\omega(\overline{x})}{\omega(\overline{(x)})}$, and semi-normal are of the form $\frac{\alpha(\overline{x}):\beta(\overline{x})}{\omega(\overline{(x)})}$

[2] a closed default doesn't contains any free variables, otherwise it is an open default.

Examples

These samples are directly taken in [David Poole, Default Logic]. *The birds can fly except baby birds* can be represented as follows :

$$
\begin{aligned}
D &= \{\tfrac{:birdsfly(x)}{birdsfly(x)}\},\\
F &= \{\forall birdsfly(x) \wedge bird(x) \Rightarrow flies(x),\\
&\quad \forall x bird(x) \wedge baby(x) \Rightarrow \neg birdsfly(x),\\
&\quad bird(Tweety),\\
&\quad baby(Polly),\\
&\quad bird(Polly),\\
&\quad baby(Keith),\\
&\quad \neg flies(Fred)\}
\end{aligned}
$$

Remember that F is the facts, given to the system or computed by some processus and D[1] are the defaults. Note for instance that we can then derive a newfact : $birdsfly(Keith)$ by default ie from D.

This description led to some conclusions : *"There is one extension that contains* $birdsfly(t)$ *for every term* t *(except for* $t = Polly$*).* $flies(Tweety)$ *can be explained using the explanation* $F \cup \{birdsfly(Tweety)\}$*. We cannot explain* $flies(Polly)$ *as* $birdsfly(Polly)$ *is not consistent with the facts. We can explain* $\neg bird(Keith)$*, using* $F \cup \{birdsfly(Fred)\}$*. We can also explain* $bird(t) \Rightarrow flies(t) \wedge \neg baby(t)$ *for every ground term* t *(except for* $t = Polly$*)."*

The same situation can be represented with another form :

$$
\begin{aligned}
D &= \{\tfrac{bird(x):flies(x)\wedge\neg baby(x)}{flies(x)\wedge\neg baby(x)}\},\\
F &= \{bird(tweety)\\
&\quad baby(Polly),\\
&\quad bird(Polly),\\
&\quad \neg flies(Fred)\}
\end{aligned}
$$

Here, the default is also normal and the description mimes the same situation but the consequences are different. We see here that the syntax is really important in this process. *"we can explain* $flies(Tweety)$ *and* $\neg baby(Tweety)$*, by assuming the default for* $x = Tweety$*. We still cannot explain* $flies(Polly)$*, and can no longer explain* $\neg bird(Fred)$*, nor* $bird(t) \Rightarrow bird(t) \wedge \neg baby(t)$ *for an arbitrary ground term* t*."* The main difference between the two representation is that we need to derive $bird(t)$ before applying the default, which

[1]Here, this is a unique normal default

led to some trouble.

We thought this sample be a good one to show the importance of checking for consistency, and the difficulties of creating a good representation considering the importance of a syntax choice.

2.2.3 Circumscription[1]

Let's now take an overview of the circumscription theory, which is another good example of nonmonotonic reasoning. Unlike default logic, circumscription doesn't suffer from the choice of the chosen representation, implying less problems is choosing the way we describe it.

This time, we will work with abnormal predicates, for the exception. The rules will be like $b(x) \wedge \neg ab(x) \rightarrow f(x)$ which is equivalent to : $b(x) \wedge \neg f(x) \rightarrow ab(x)$.
The main idea of this formalism is to minimize the domain in which all formulaes are consistent.

In fact, in our world, there is a lot of interpretation. We will choose the smallest possible interpretation of our model that is consistent with the rules and facts of which the model is composed to represent the circumscription. Note that we can also have more than one smallest interpretation.

Figure 2.1 below represents different interpretations I_i and the set of interpretation in a model A. We can see in blue the circumscriptive effect that takes the smallest interpretation of Mod(A). To be more precise, the lines in the drawing represent all comparable relations between representations (down to up). For instance, $I_6 \geq I_5$ where $\geq$ is the given comparison[2] function as we'll see below.

[1] Based on note[LIF94]
[2] partial order

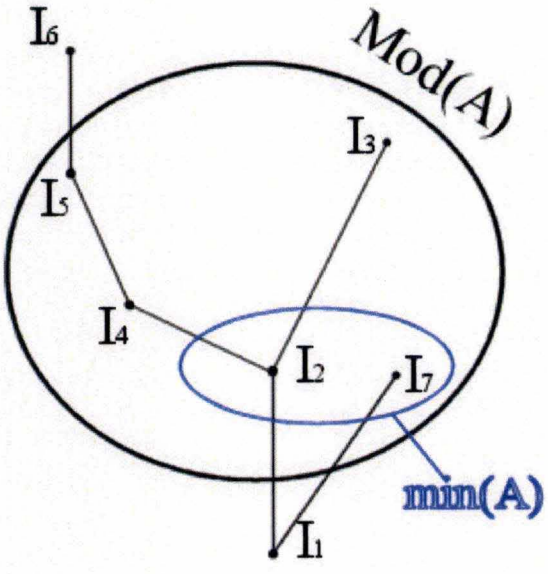

FIG. 2.1 – Model and Interpretation - circumscription.

Semantic

Let's begin with semantic of the circumscriptive interpretation.
Interpretation is : $I : \mathcal{P} \rightarrow (\mathcal{D} \rightarrow \mathbb{B})$.

We first need explain the partial order between interpretation :

$$\begin{array}{ll} I_1 \leq I_2 & \triangleq \quad I_1(ab) \subseteq I_2(ab) \\ \wedge\ I_1(T) = I_2(T) & (T = type) \\ \wedge\ I_1(p) = I_2(p) & (p = predicate) \\ \wedge\ I_1(f) = I_2(f) & (f = function) \end{array}$$

minimal model is then $min(A) = \{m_1 \in mod(A) |\ \nexists\, m_2 \in mod(A) | m_2 \leq m_1\}$[1]

[1] $I_1 < I_2 \equiv I_1 \leq I_2 \wedge I_2 \not\leq I_1$

Syntax

A model can easily be represented by a unique second-order logic. If we have two interpretations representing the same model, we need to take the interpretation which contains the least restriction. If we have two distinct interpretation I_1 and I_2, let be ab_1 and ab_2 the only difference between the two interpretations. The translation of minimal models into the second order logic is as following, taking A as a simple logic sentence, with the default predicate ab :

$$(A[ab := ab_1] \wedge \nexists ab_2.A[ab := ab_2] \wedge ab_2 < ab_1)[ab_1 := ab]$$
$$\equiv [A \wedge \nexists ab_2.A[ab := ab_2] \wedge ab_2 < ab] \quad \textit{(minimal models definition)}$$

where : $ab_1 \leq ab_2 \triangleq \forall x : T.ab_1(x) \rightarrow ab_2(x)$
$ab_1 < ab_2 \triangleq ab_1 \leq ab_2 \wedge ab_2 \not\leq ab_1$

or in contracted version, $CIRC[A, P] = A(P) \wedge \neg \exists p.[A(p) \wedge p < P]$

Resolution

These formulaes can be resolved by simplification, then with appropriate choice of a substitutive predicate p, which need to be the most general. The results are often very intuitive, but not all the time.

The example of $A \equiv Q(x) \supset P(x)$ leads to

$$A' = CIRC[A, P] = \forall x[Q(x) \equiv P(x)]$$

which is a good intuitive solution, as the solution domain, the interpretation, of A' are always true in the starting theory A, and moreover, this is the smallest set where this property is true.

Surely due to this basic case of circumscription is too specialized for most of applications, the way we limit our workspace without changing anything else in the model. An extended definition has been created to face this limitation. We can then have a possible change of external functions or predicates(called $z_1, ..., z_n$ denoted by Z) :

$$CIRC[A, P, Z] = A(P, Z) \wedge \neg pz.[A(p, z) \wedge p < P]$$
with arity conservation for p and z, regarding P and Z.

Theory

Like in default logic formalism, an idea of a circumscriptive theory can be drawn :

A circumscriptive theory is defined by a set Γ of sentences, called the *axioms* of the theory, and a set Δ of expressions of the form "$circ\ P\ var\ Z_1, ...Z_n$". That form of expression means that circumscription will be used on predicate P with the constants $Z_1, ..., Z_n$ varied[1]. This Δ is called *policy declarations.*

A model of a circumscriptive theory(Γ, Δ) is any model of Γ that is minimal regarding $\leq^{P;Z_1,...,Z_n}$ corresponding to the policy declarations. We can then access to theorems of (Γ, Δ) that are sentences which are true in all its models.

Computation

Surely, there is also a lot of propositions, lemmas and theorems that are used to accelerate the process of resolution without this kind of intuition we need to progress. To this end, you can refer to [LIF94].

The Bird Example

Once more, we can describe a representation of the bird example that we met twice above. Suppose that we want to represent the problem we had in simple logic chapter, it should give a system as :

$$\Gamma = \begin{cases} bird(x) \wedge \neg Ab(x) \supset fly(x) \\ bird(Mike) \\ fly(Mike) \end{cases}$$

with the circumscription $\Delta = \{circ\ Ab\}$[2]

[1] with the restriction that Z_i is not containing P

[2] A more interesting example would have more than one circumscribed predicate, with varying constants. See [LIF94] for more.

2.3 Belief Revision

Belief revision could not be explained better than Mary-Anne Williams [WIL95] just as we couldn't explain the AGM paradigm better than its inventors Alchourrón, Gärdennfors and Makinson [AGM85]. Our objective in this paper is to present their principles in a general overview. In fact, belief revision has been used only in its general principle. We did not play the limit of this formalism for our application.
To this end, we'll try, once more, to sketch a simple overview or intuition of this theory. The essential requirements to understand its power and usage will be explained. The link between previous nonmonotonic reasonings that we met in previous sections and belief revision will be underlined. This is important to fix our idea for the real application.

2.3.1 The Core

Belief revision systems offer different mechanisms to manage new entries of knowledge into a knowledge base. It's not only a method to develop but also to verify the consistency of its incremented base. Indeed, some new information could be in conflict with some axioms or consequence of axioms in this world.
The great interest of this formalism is based on minimal change principle. This one is relatively simple to understand : we try to suppress only the least important axioms, while the new information added on the system is considered as more important than any other belief : a true belief.

We can see it in a simple logical example. Starting with $\{I_i\}_{[1..3]}$ and adding I_R, we conclude to an inconsistence :

$$\left\{ \begin{array}{ll} I_1 : & A \\ I_2 : & B \\ I_3 : & A \wedge B \rightarrow C \end{array} \right\} \quad \wedge \quad \{\ I_R :\ \neg C\ \} \quad \rightarrow \bot$$

To remove the inconsistency, we need to suppress I_1, I_2 or I_3 on top of I_R[1]. It could be interesting to suppress more than one of the I_i leaving the base maybe less rich but surely more stable.

[1]because I_R is a consequence of the conjunction of the I_i

The main question is then to know how define an ordering between beliefs. In fact, it could be more efficient to withdraw 2 less important axioms instead of one more important. For this, two methods have been proposed :

- define a possibility value for any axioms or rules called entrenchment.
- use a system of sphere representing the different possible(distinct of a probabilistic viewpoint) world extending from the "more possible" to the "less" one.

Let's take the following example system :

	Axioms/rules	*Entr.*	Added axiom(considered true)
I_1	A	$0,99$	
I_2	B	$0,99$	
II_1	$A \rightarrow C$	$0,3$	$IV : C \wedge D$
II_2	$B \rightarrow C$	$0,3$	
III_1	$\neg D$	$0,8$	

The most logical conclusion is to choose to withdraw[1] II_1 and II_2 instead of III.

To develop the theory, we need to define some operators that take a knowledge base $\mathcal{K}_{\mathcal{L}}$and a symbol of the language $\mathcal{L}$ (representing an axiom, the belief) and provide a new knowledge base.

Here is a brief description of actions before the formal definition :

withdrawal : Suppression of certain axiom, without any other verification.

contraction : Suppression of an axiom and of all of the rules implying it.

expansion : addition of a new axiom in the theory base

revision : addition of a new axiom with the respect of the system consistency.

These actions constitute the AGM paradigm's kernel, created *for modelling ideal and rational changes to repositories of information under the principle of minimal change.*

To be more precise in the definition, we should implement these functions with respect to the minimal change principle as defined in [WIL95] :

Expansion

$$\begin{array}{rccl} + : & \mathcal{K}_{\mathcal{L}} \times \mathcal{L} & \longrightarrow & \mathcal{K}_{\mathcal{L}} \\ & (T, \varphi) & \longrightarrow & T_{\varphi}^{+} = C_n(T \cup \{\varphi\}) \end{array}$$

[1] with the most standard rule

Expansion is the simplest operation[1]. This is a normal monotonic operation to add a belief in the theory base. We have a consistent theory without other changes *iif* $\neg\varphi \notin T$.

Contraction

$$
\begin{array}{lccl}
- : & \mathcal{K}_{\mathcal{L}} \times \mathcal{L} & \longrightarrow & \mathcal{K}_{\mathcal{L}} \\
 & (T, \varphi) & \longrightarrow & T_{\varphi}^{-} \quad \text{such as :}
\end{array}
$$

$$
\begin{array}{ll}
(^-1) & T_{\varphi}^{-} \in \mathcal{K}_{\mathcal{L}} \\
(^-2) & T_{\varphi}^{-} \subseteq T \\
(^-3) & \text{If } \varphi \notin T \text{ then } T \subseteq T_{\varphi}^{-} \\
(^-4) & \text{If } \nvdash \varphi \text{ then } \varphi \notin T_{\varphi}^{-} \\
(^-5) & T \subseteq (T_{\varphi}^{-})_{\varphi}^{+} \qquad (\textit{recovery}) \\
(^-6) & \text{If } \vdash \varphi \leftrightarrow \psi \text{ then } T_{\varphi}^{-} = T_{\psi}^{-} \\
(^-7) & T_{\varphi}^{-} \cap T_{\psi}^{-} \subseteq T_{\varphi \wedge \psi}^{-} \\
(^-8) & \text{If } \varphi \notin T_{\varphi \wedge \psi}^{-} \text{ then } T_{\varphi \wedge \psi}^{-} \subseteq T_{\varphi}^{-}
\end{array}
$$

An interesting point of these properties is the postulate ($^-5$) which says, according that $\varphi \in T$ and with the previous four postulate, that $T = (T_{\varphi}^{-})_{\varphi}^{+}$. This is a powerful feature because it says that *no more information is lost that can be reincorporated by an expansion with respect to the explicit information contracted, that is, if we contract φ and then immediately replace it using expansion then we obtain the theory we started with. Intuitively then, this postulate forces a minimal amount of information to be lost during a contraction*[2]. The withdrawal function fulfills all the postulate but the recovery ($^-5$).

Revision

$$
\begin{array}{lccl}
* : & \mathcal{K}_{\mathcal{L}} \times \mathcal{L} & \longrightarrow & \mathcal{K}_{\mathcal{L}} \\
 & (T, \varphi) & \longrightarrow & T_{\varphi}^{*} \quad \text{such as :}
\end{array}
$$

$$
\begin{array}{ll}
(*1) & T_{\varphi}^{*} \in \mathcal{K}_{\mathcal{L}} \\
(*2) & \varphi \in T_{\varphi}^{*} \\
(*3) & T_{\varphi}^{*} \subseteq T_{\varphi}^{+} \\
(*4) & \text{If } \neg\varphi \notin T \text{ then } T_{\varphi}^{+} \subseteq T_{\varphi}^{*} \\
(*5) & \text{If } T_{\varphi}^{*} = \perp \text{ then } \vdash \neg\varphi \\
(*6) & \text{If } \vdash \varphi \leftrightarrow \psi \text{ then } T_{\varphi}^{*} = T_{\psi}^{*} \\
(*7) & T_{\varphi \wedge \psi}^{*} \subseteq (T_{\varphi}^{*})_{\psi}^{+} \\
(*8) & \text{If } \neg\psi \notin T_{\varphi}^{*} \text{ then } (T_{\varphi}^{*})_{\psi}^{+} \subseteq T_{\varphi \wedge \psi}^{*}
\end{array}
$$

In other words, we can see that revision is nonmonotonic. We can also conclude that revision is a combination of expansion and contraction.

---

[1] except withdrawal which is trivial

[2] In our opinion, this is an essential postulate to take into account while implementing this function, which is probably not as easy as it seems.

The AGM paradigm's creators have also demonstrated the relationship among the different functions :

- The *Levi Identity* : If $^-$ is a contraction function and $^+$ is an expansion function, then we can then define a revision function by
$$T^*_\varphi = (T^-_{\neg\varphi})^+_\varphi.$$
- The *Harper Identity* : If * is a revision function then we can define the contraction function $^-$ by
$$T^-_\varphi = T \cap T^*_{\neg\varphi}$$

2.3.2 Epistemic Entrenchment

In order to define an ordering between the different rules and axioms that we believe, we need to define a specific Epistemic Entrenchment Ordering(EEO). This can be done by implementing some functions that comply with the following definitions :

An Epistemic Entrenchment related to a theory T of $\mathcal{L}$ is any binary relation $\leq$ on $\mathcal{L}$ satisfying the conditions below.
(EE1) if $\varphi \leq \psi$ *and* $\psi \leq \chi$, *then* $\varphi \leq \chi$*(transitivity)*
(EE2) for all $\varphi, \psi \in \mathcal{L}$, *if* $\varphi \vdash \psi$ *then* $\psi \leq \psi$
(EE3) For all $\varphi, \psi \in \mathcal{L}$, $\varphi \leq \varphi \wedge \psi$ *or* $\psi \leq \varphi \wedge \psi$
(EE4) When $T \not\perp, \varphi \notin T \iff \varphi \leq \psi$ *for all* $\psi \in \mathcal{L}$
(EE5) if $\psi \leq \varphi$ *for all* $\psi \in \mathcal{L}$, *then* $\vdash \varphi$

Indeed, if $\varphi \leq \psi$, then we say ψ is at least as entrenched as φ.
We define $\varphi < \psi$, as $\varphi \leq \psi$ and not $\psi \leq \varphi$ (as often in logic). If $\varphi \leq \psi$ and $\psi \leq \varphi$, then we say φ and ψ are equally entrenched.

- (EE1) is a simple property of transitivity.
- (EE2) says that the consequent is at least as entrenched as the antecedent. For example, we have φ that entails $\varphi \vee \psi$. It's essential that we have more conviction of $\varphi \vee \psi$ than φ.
- (EE3) combined with the precedent two axioms says that a conjunction is ranked at the same level of its least ranked conjunct.
- (EE4) tells us that the sentences not in the theory are minimal.
- (EE5) says that tautologies are minimal.

All these properties need to be followed to describe a correct ordering. They define a class of binary functions where we could get the needed characterization of any rules between them. For any epistemic entrenchment ordering, we can define the set *cut* :

$$cut_\leq(\varphi) = \{\psi : \varphi \leq \psi\}$$

The set $cut_{\leq}(\varphi)$ contains all those sentences that are at least as entrenched as φ. An important property of an epistemic entrenchment is being a total preorder of the sentences in $\mathcal{L}$ such that :
If $\leq$ is an epistemic entrenchment, then for any sentence φ,$cut_{\leq}(\varphi)$ is a theory.

From this point, we can develop further theories using several theorems given by the AGM's authors. Without going into details ([WIL95]), they serve to :

- give the finiteness of the representation of an existing entrenchment.
- prove the existence of an entrenchment for each contraction function.
- prove the existence of a specific contraction function for each entrenchment.
- prove the existence of an entrenchment for each extraction function.
- prove the existence of a specific extraction function for each entrenchment.
- give the number of extraction an contraction functions for a given theory.

Implementing Entrenchment

Two major problems appear while when trying implement entrenchment.

The first lies in the need to support iterated revision. The basic transformation defined in the AGM paradigm take an epistemic entrenchment ordering coupled with a sentence and transforms it into a theory at which point the ordering is lost. That's why repeated revision iterations can't be done.

The second problem is the typical infiniteness of the ranking of sentences[1] by an epistemic entrenchment, which is a well-known problem in computer implementations.
As [WIL95] says, we can define the relationship between the formal epistemic entrenchment ordering and the implemented partial entrenchment ordering.

Finite Partial Entrenchment Rankings[2]

The definition below can be taken to understand the use of a finite partial entrenchment ordering(FPEO), graduating the content of a finite knowledge

[1] due to infiniteness of the set of sentence contained in the base.
[2] FPER

base, according to its epistemic importance. In practise, it's a mapping from any sentence to a rational number.
A finite partial entrenchment ranking is a function B from a finite subset of sentences into the interval $[0,1]$ *such that the following conditions are satisfied for all* $\varphi \in$ dom(B) :

$(PER1)$ $\{\psi \in dom(B) : B(\psi) < B(\psi)\} \nvdash \varphi$
$(PER2)$ $If \vdash \neg\varphi, then B(\varphi) = 0$
$(PER3)$ $B(\varphi) = 1 if and only if \vdash \varphi$

A qualitative use can be observed as we graduate a sentence relative to another. Conversely, we observe another use which is quantitative as we give the specific entrenchment for a sentence corresponding to a necessity, a kind[1] of probability. The system will then compute the other sentences from the numeric value.

We can then separate two kinds of belief, the explicit and the implicit ones, that come from system derivation. As usual, the first is the described theory defined by :

Given a finite partial entrenchment ranking B, the explicit information content represented by B, denoted by $exp(B)$, is $\{\varphi \in dom(B) : B(\varphi) > 0\}$. The implicit information content represented by B, denoted content(B), is $Cn(exp(B))$[2]. A lot of convenient rankings are coherent with a starting base. We'll choose the minimum possible degree of entrenchment.

Once we have a ranking, it should be necessary to create an epistemic entrenchment ordering from this ranking. To this end, we define how to assign a degree of acceptance to implicit information :

Let φ *be a nontautological sentence. Let B a finite partial entrenchment ranking. We define the degree of acceptance of* φ *to be :*
$degree(B, \varphi) =$

$$\begin{cases} j | j \geq i \text{ where } \{\psi \in exp(B) : B(\psi) \geq i\} \vdash \varphi & \text{if } \varphi \in content(B) \\ 0 & otherwise \end{cases}$$

The simplest way to compute the degree of an expression lie in a straightforward top down procedure : We can take all sentences from the highest degree[3] and try to derive the expression. If we succeed, the reached degree is

[1] Entrenchments don't work as with probability.
[2] Cn is a logic function often used giving the set of all possible conclusion of a theory. Refer to [MAK94] for further explanation for example.
[3] Remember that tautologies are degree 1 and inconsistency a degree 0

set, else we try with the degree just below and so one until reaching 0.

Then the theorem below finishes to show how the FPER generates an EEO using the degree of acceptance :
Let B a FPER an φ, ψ be sentences.
Define $\leq_B$ by $\varphi \leq_B \psi \iff \vdash \psi$, or $degree(B, \varphi) \leq degree(B, \psi)$.
Then $\leq_B$ is an epistemic entrenchment ordering related to $content(B)$.

Iterated Revision

The lack of a policy to support iteration of change function in the AGM paradigm can be resolved in practise. In fact, this omission is useful to keep all different implementations available. As we want to modify a ranking, we need not only the sentence but also a degree of acceptance which is in [0,1[. The policy for change quoted above is linked with a function of minimal change. The computation of a minimal change function can vary with respect to our goal. The exact definition of the adjustment[1] function is a bit long but the concept is understandable enough to imagine without formalizing. Adjustments define change functions for theory bases, rather than logically closed sets of sentences as in the previous section.

Intuitively, a (φ, i)-adjustment of B involves minimal change to B such that φ is accepted with a degree i. In other words, to keep the system consistent, all sentences ψ in B will be reassigned to the closest number of B(ψ) chosen in the adjusted partial entrenchment ranking obtained under the guiding principle that if we reduce the degree of the new accepted φ to i, we will also reduce the other relative sentences, as in the contraction process.

We could continue to identify some of propositions comparable to Harper and Levy identity but we don't think it's essential understanding. Our reference paper describes a good example for special adjustments that you could refer to, for a practical description. Anyway, different adjustments method will be overviewed in the next subsection.

Well-known Adjustments

Some adjustment definitions can be quoted for the Saten System described in the next chapter. Saten implements Standard AGM, maxi-, hybrid, global, linear and quick adjustment as defined in [SADJ].

[1] You can check it in [WIL95], the source of all this section

The Standard adjustment is based on the standard epistemic entrenchment construction for belief revision. It seems that it proceeds from the top of the ranking and moves down it rank by rank until it finds inconsistence. Then, any belief of the same rank or below is discarded.

Maxi-adjustment proceeds from the top of the ranking and moves down it rank by rank. At each rank it deletes all the beliefs that are inconsistent with other belief at that rank and above.

These are the adjustments we used to work with. The other adjustment are described in [SADJ].

This concludes our theory chapter.

Chapitre 3

Systems

Belief Revision is a good approach for artificial intelligence. Our goal is to run it on a specific robot, already used in international RoboCup competition. In fact, we would like to have some kind of dynamic behaviour for a robot in all possible environments. Basic weak[1] AI vision, like the actual development for Aibo, is not far enough to achieve this goal. The need to recompile and build another case for any special action for the fake being is reluctant and time-consuming. The robot is not as autonomous as we hope. The good feature of Belief Revision is the modification of the way to behave and think in real-time.

The description of the system implemented for Aibo below could be used on another platform as we essentially work with Java. In the next section we will focus on the description of all problems encountered during development. All the code is fully documented and presently available on [MYWWW][2].

3.1 System Implementations

The system we made is based on a client-server principle. First of all, we tried to have a separate code method to let us improve one part or another when needed. Thus we'll speak about all the different views in the next subsections.

[1] In short, weak AI corresponds to a particular code for any action, using hardware properties to be a very effective way to work. Strong AI is oriented in simulating human brain, deduction or behaviour(e.g. Neural Networks,...)

[2] the system is still improving

3.1.1 Systems Used

Two major softwares have been integrated to complete this part of the system, constituting the core of process.

The first software is the belief revision system of the Queensland University of Technology(Australia) as known as AIFS[AIFS]. It's a belief revision system based on three different theorem provers : robdd solver, buddy solver and sicstus solver. Robdd is a Reduced Ordered BDDs solver which follows SAT technics. Buddy is another BDD solver[1]. Sictus is a prolog distribution providing some theorem provers as prolog programs. We choose not to go further in defining theorem provers. The AIFS system implements four different adjustments as AGM, maxi, maxi approximated and linear change functions.

In other hands, the second software is SATEN[SATEN], developed in NewCastle university(Australia) and supervised by Mary-Anne Williams. This one is based on a proprietary prover called vader, also developed in NewCastle. The Saten system is able to perform nonmonotonic reasoning, possibilistic reasoning, and hypothetical reasoning. The available adjustment are AGM adjustment, hybrid, linear, maxi, nebels and speedy. We will discuss the differences later. Both systems were coded in Java the entire system.

3.1.2 Believer

This program was needed to link with a human-handable interface for this extraction-revision systems for belief revision. In fact, the hardest task consists on the translation from one system to the other as they are not compatible.

BelieverCommandLine

The Believer contains a text input interface coded in Java.
See Figure 3.1 below.

[1]http ://sourceforge.net/projects/buddy

The command line allows users to :

- Create and Run a Revision Engine in the specified prover and adjustment.
- Stop and Kill a Revision Engine.
- Load a previously saved theory
- Save a given theory
- Revise a new sentence with a degree into the actual knowledge base
- Show all the knowledge base information
- Show all ranked sentences of the knowledge base, with their explicit entrenchment degree.
- Ask for degree of a given sentence or expression.
- Navigator-like transition as Next-Previous to undo/redo actions.

The command line checks for all possible errors as parsing errors, illegal entrenchment values, incorrect save names ...

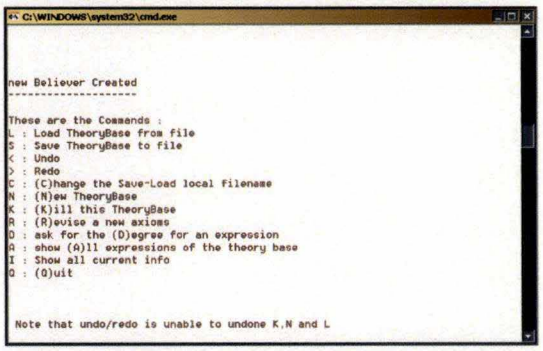

FIG. 3.1 – Believer - Command Line.

3.1.3 Core

The internal system manages also other features :
- Implementation of all command line tasks.
- Synchronized thread management for different tasks[1].
- Automatic conversion of input format for SATEN into AIFS format.
- Separated implementation of command line system to be overriden in further development.
- Error and confirmation message management using own protocol[2].

3.1.4 WebServer

The WebServer consists on a socket server doing the factory work for any incoming socket. It links all incoming requests to the Believer when available. The main features are :
- Stack of multi-threaded external request
- Auto resizable thread pool for minimizing memory allocation and latency.

3.1.5 WebClient

The WebClient is the simple client layer providing socket opening. It just invoke a new socket when BeliefViewer asks for it and forward the requests.

3.1.6 BeliefViewer

BV is another great part of the whole work. It's task is to provide a more pleasant GUI client application to access the intelligence server.
The program is separated into two tabs. The first is "administrative tools" while the second is "theory base".
Some "Printscreen" are available at the of this section : Figure 3.2 and 3.3.

Administrative tools

The window is separated between different semantic zones :

Communication Options
- Server IP and Port
- Local IP and Port

God-like Options (Management of Revision Engine)

[1] useful for serving or for multitask command line systems if implemented
[2] which will be used by the client-server

- Set of different provers (robdd, buddy, sistus and vader)
- Set of different adjustments, depending on the prover selected
  - Standard AGM, Maxi, Maxi approximated and Linear for the first three provers
  - AGM adjustment, Hybrid, Linear, Maxi, Nebels and Speedy for Vader
- A time value for approximation for all prover but Vader.
- Creation or Destruction of Revision Engine Button depending on context.
- "Morphing" script button : After a dialog with dialog box, permits to convert a given theory base

Saving Options

- Text input for filename
- Load and Save button.

Info

- Refresh : reload information from the server
- Server Status : provide information of the running state of the server

This description is not complete : a lot of case base interface changes have been implemented to hide, focus or disable buttons in different contexts.

Theory Base

The second tab, accessible when the server is running a revision engine, contains the knowledge base. It permits interaction with sentences.

Running belief

- The knowledge base is represented by a simple table which contains
  1. Ranking
  2. Entrenchment or degree
  3. Sentences or expressions
- Different beliefs can be selected to be used with the action part.

Actions : Queries to the server

- Expression and Entrenchment input
- Choice of actions with combo-box
  - Revise a given expression with a certain degree
  - Ask for the degree of a given expression
- Dismiss button (when the option is activated) to revise an expression to 0 degree.
- Navigation control for undo/redo actions (Next/previous)
- Refresh

Note that all feedback, local or from the server are handled by a dialog box, same as confirmation of actions.

3.1.7 Definition

We followed for our development an interface coding to have some monitoring different of protocols, string definitions, parameters between different parts of the code. A compilable Java file called "Definition.java" groups all parameters in different interfaces for each domain (Believer, Client/Server, prover options ...). It permits us to have a revision system.

3.1.8 Log

All different parts described above are always logged. It provides a method to trace any bug and a backtracking method to reconstruct theory base if the server crashes.

3.1.9 Aibo Client

Aibo dog works with a special processor running a specific dedicated operating system called Apertos[1]. The memory of this puppy is stored in a memory stick[2], and it needs a specific configuration of different files. For example, if we need to work with FTP client, we can put in some special directory a program given by Sony to have the system running. We could also code our own version of the program, starting from scratch or completing the Sony version, which is very limited.

To this end, we need to code with a special version of C++ called Open-R[OPENR] designed for Apertos. To do so, we need to install some add-on to Linux installation, to have a GCC[3] compiler for Open-R. Once compiled, we can access to a special directory to have native code to put on Aibo. The amazing stuff with this method is that we just need to put some different programs all together in the memory stick to have them all running together (if there is no unsolicited side effect). For inter-object communication, a special interface has been provided.

---

[1]or Aperios, http ://www.csl.sony.co.jp/project/Apertos/techpaper.html (Japanese)
[2]http ://www.memorystick.org
[3]GNU C Compiler

To contact the intelligence server, we have modified the soccer team code. The example we develop for the simple demonstration in Australia is a request to play. The dog contact his "brain" and asks if it wants to play. In case that the server computes a negative answer, after using belief revision, the dog sit down and blinks some leds. In other hand, if it want to play, it stands up and wags his tail, turning on other leds. As we did not have enough time in Australia, we finished the development for the dog in the lab, keeping the belief revision application to be done in Belgium, on the delocalized server.

The code is available on a website but is not added in annex to this thesis for three reasons :

- A raw code is not pleasant to read nor interresting.
- A Javadoc is better if we use HTML link.
- The program is still being improved (with 90% completed javadoc).
- Printing more than 5000 codelines(only our new code) for swaggering is a bit silly.

We'll see in the next section all the problems encountered, mostly in the Aibo implementation.

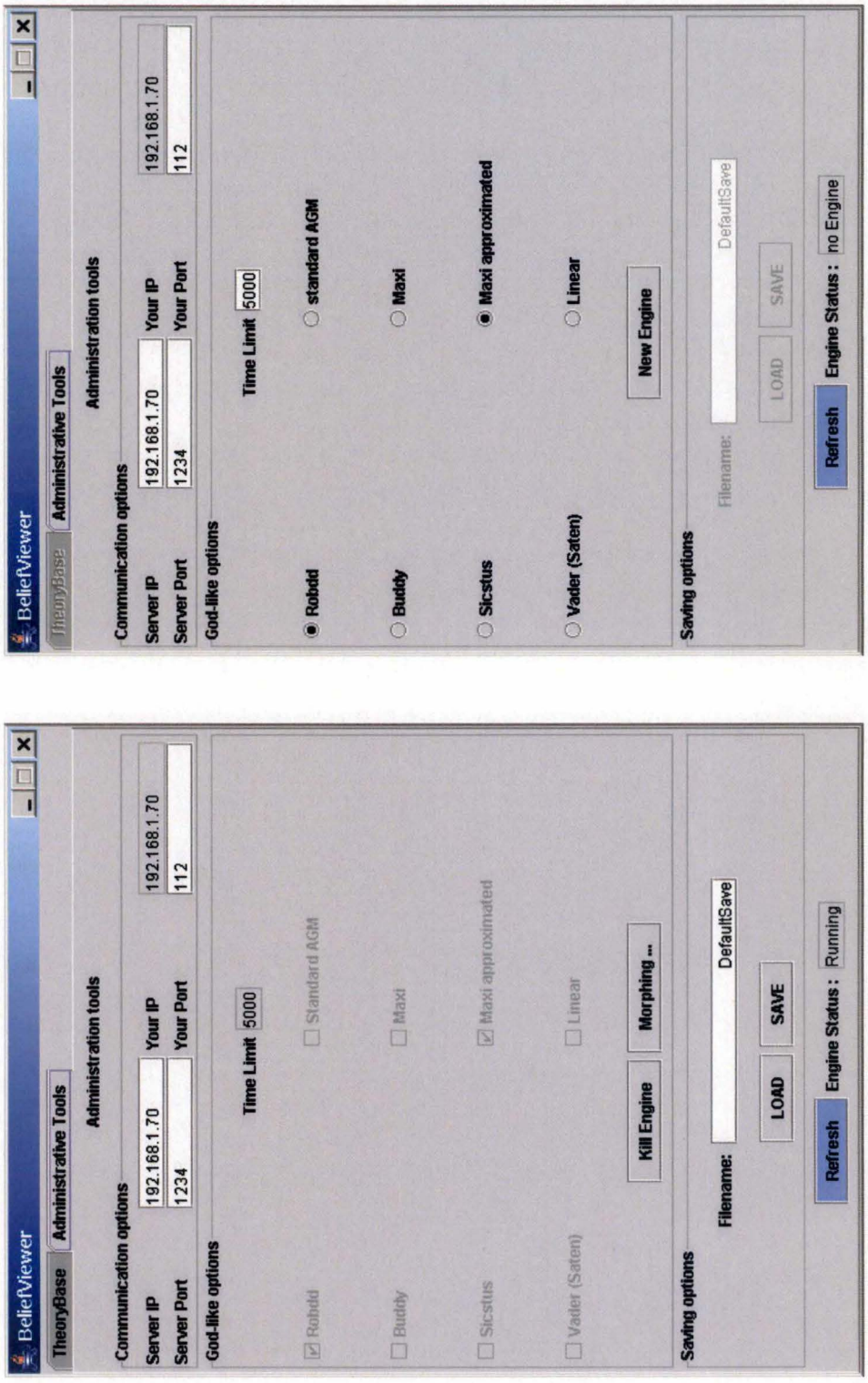

FIG. 3.2 – Belief Viewer - Administration tab.

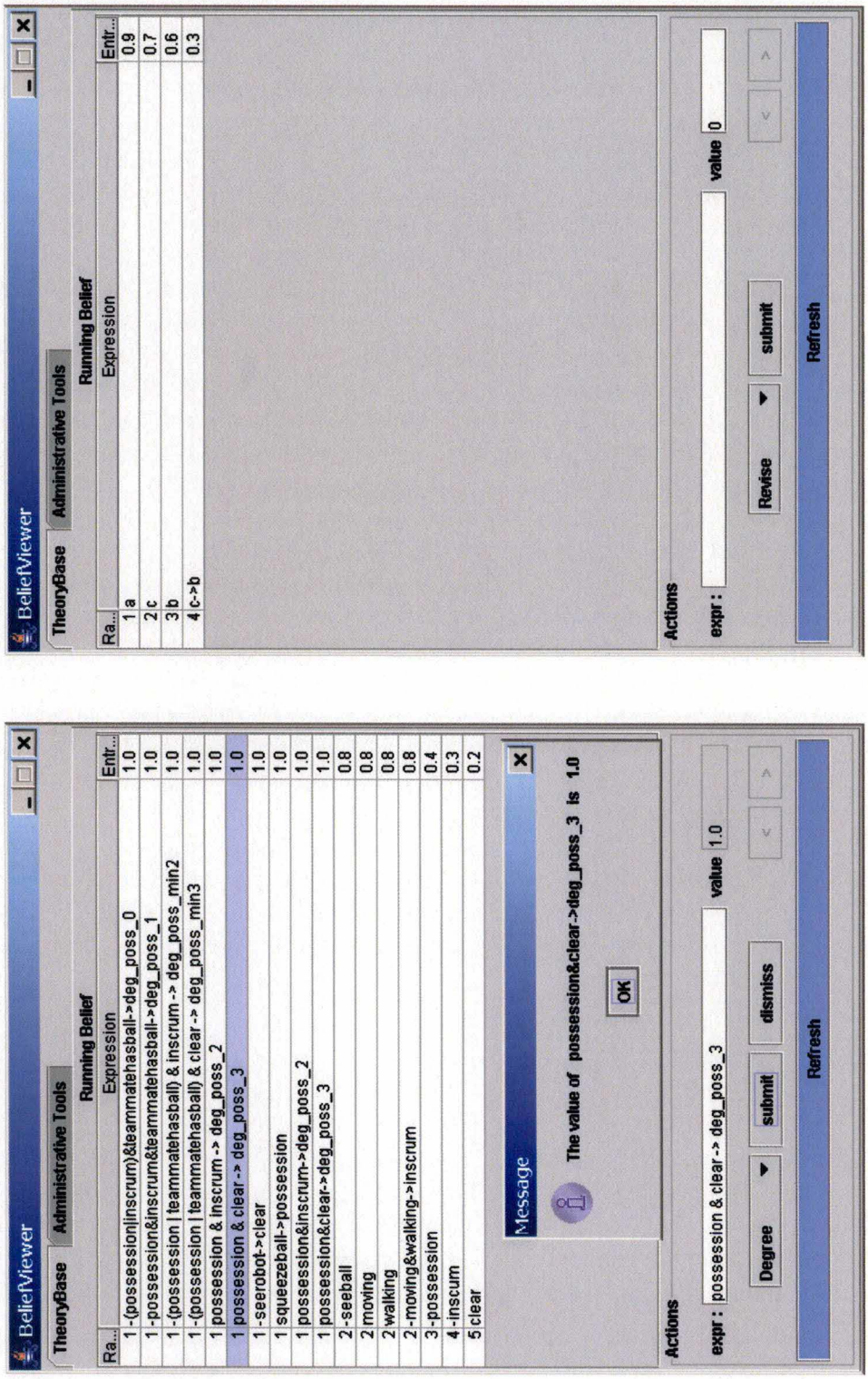

FIG. 3.3 – Belief Viewer - Theory Base tab.

3.2 Problems Encountered

We'll start with architecture problems and continue with coding problems, in parallel of the description given above, in the previous section.

3.2.1 Architecture Problems

Our goal is to run the intelligence system on Aibo to have a completely independent robot. The first trial was to look for a conversion from java to C++ for Saten or any other belief revision system. In fact, we conclude that will be a hard task in such little period of time. SATEN or AIFS have been time-consuming developments and bugs have been discovered and fixed after years. For the moment, we reject the possibility to rebuild a similar program in Open-R which is a very young language with some unreported side effects. We don't want to have a buggy program to start working with Aibo.

The second trial was to convert the AIFS program from Java to C++ using some Java interpreter in C++ as JNI[JNI] or direct compiler in Apertos as gcj[GCJ] for Linux or other commercial products. After weeks of research, we did not find any useful tools of this kind. We hope that these tools could be developed as for Lego robot[1]. The same research have been made deeper by other students or scholars who didn't succeed.

We conclude with our last possibility : the client-server architecture. This method was proposed in the beginning of the development but we put it aside due to the delocalizated viewpoint. Indeed, this way to do is restrictive because we need always a network connection, a server responding and we don't want any interaction with other computers or human brain in the first place. Anyway, this choice provides us other possibilities to work with. We will discuss about it later.

3.2.2 Architectural Problems

Believer

We had first a lot of trouble with wrapping SATEN in AIFS formalism. The differences between the two software led us to serious incompatibilities. We had to study the SATEN code to determine the formalism due to a lack of Javadoc and then make a wrapper to convert one system to another. The methods for revision engine creation, null theory base manipulation and others

[1] http ://mindstorms.lego.com/

were also a challenge. It would had been smarter if the specification in both languages were the same. We think the creation of a good documentation of the system workflow is essential to continue to work with these systems.

Aibo

The main problems occured here. This information could be useful for any student who wants to work with our system. Learning C++ is just a prerequisite in this process.

First of all, the inter-object communication had some trouble to work properly. The problem was not fixed but it seems to come from the memory problem we present below. We change our direction using one big object, like the one created in the UTS lab. This version isn't probably the same anymore

The second problem came from TCP connections : the TCP code was only stack server code, which is a little limited as it cannot access a specific IP address and port. This specific architecture used in the lab has been understood and debugged with the few added included information. The missing TCP client method were developed with respect to the team code. This respect was the hardest task in this process.

Third, a big problem was discovered probably with memory allocation. The repeated crashes were surely due to a dirty memory access without allocation in some part of the team reference code. We tried to contact Sony to understand the cause.
This problem has consequences for the visibility of our code : only static memory allocation could be used(no *new* or *dispose* could be executed.). In fact, the proper code is still commented out in the final version, unless the problem has been fixed.

The last problem was the long process to compile, copy on memory stick, run the aibo, use a very limited debugger after recopying the memory stick to pc ... These manipulations took a very long time in each trial. It could be useful to have an emulator of apertos and a good debugger.
The difficulties of managing motors and movement for the Aibo was not the point, because we understood and used a part of Suzanne Grell's code.

Chapitre 4

Applications

4.1 Architecture

The Client Server architecture isn't the perfect way to implement our brain for Aibo, but we still can find good applications for this way of working. The next subsection of this part will introduce some of these aspects.

4.1.1 Telepathy

In former development for RoboSoc competition, UTS Unleashed team always used to work with Aibo alone. All information of the world model, intelligence, views ... were accessible via TCP connection to Aibo, who was serving all this information. This way to work is better for rarely accessed information like view, which is only useful for development or debugging in this case : few connections are then needed to Aibo.

In other hand, systems like world model or intelligence and belief could be used by most of the external actors. We mean that other Aibo mates could use this information for their own reasoning. In such case, a lot of connections are likely to exist. This implies that our puppy could be overwhelmed by incoming requests. Our solution, another server just for belief is a good solution because Aibo's network use wifi connection which is slower than wire networks. All the more so when the connection bandwidth drops during competition due to the number of dogs present.
The possibility to read in the mind of any puppy without decreasing its capacity is a strong reason to use a dedicated server.

4.1.2 The hive

We assert that the human brain is divided in two parts. The left side is reasoning and rationality and the right side is artistic. We can also divide the Aibo's mind in three parts : one for physical rationality postulates, another part for private feelings and thought, and we could add a third part for some mass information described as below. We could implement this using belief revision.

The private part, which develops the behaviour, emotions, character, mood can be implemented on any platform.
The development of delocalized server for the Aibo's mind is useful for the two other parts.

Tautology Server

We could dedicate a special server, containing all well-known tautologies and mathematically proven physical concepts. It could be used by all other dogs ie. all belief revision server from the other mates. It can also be implemented by simple logic or even by semantic web. But BR is also a good solution for this, if it works better with certain information.

Aibo Community

For empirical reasoning, we could also use the general formalism known as belief revision. The difference with the emotional way to believe is that we could use a global mind, like in our "best" sci-fi movies or like the way some insects works in community (ant, bee,...). Each different actors connected to this special network could improve a global pool of experience, the latter concluding itself to an empirical logic.

4.1.3 Save and Crash

Another important feature of this dedicated server is the crash safety it implies. Indeed, saving a behaviour, a belief, a dog state is possible, with more power than with the little memory of Sony Aibo. If an Aibo puppy crashes for any reason, its "brain" is still working on another site. The dog body and processor could be replaced by another mate, with the same information and behaviour.

4.2 Samples

Three useful examples have been developed for this paper in belief revision. First of all, we managed to make some AI improvements for RoboCup competition. The second example was psychological behaviour for Aibo. The last one is a trial to implement a specific behaviour which could be used as AI improvement in entertainment, more precisely for a video game.
The difficult part lies in using exceptions and mechanisms of belief revision in practical situations without any academic example.

4.2.1 Case Based Reasoning

As described in [CBR], UTS Unleashed tries to use a case based reasoning for their team management and individual Aibo behaviour. Here is a brief description of this viewpoint.

In a soccer game, we can develop some strategies to cope with particular situations. For example, one obvious basic strategy could be to force an attack with 2 or more players in a specific instance. The case could be when all opponents are unavailable (too far to intercept) and when the 2 players own the ball. There are certainly more complex pre-configured strategies, but that's not our goal here[1]. We need to compare the actual current situation with the given case database. The problem of this comparison with these global cases is to define how to compute which situation is the nearest.

To explicitly define a situation, we just take some useful characteristic of the situation. As characteristics, we have some physical or abstract concepts that describe one dimension of our situation. Indeed, our complete[2] description will be represented by a conceptual space[3]. In a sense, it approaches the notion of a topological space. Then we need some kind of measurement to compute how far current situation is from each reference one, and choose the nearest.

---

[1] a base case strategy will in fact develop step by step to the better solution with more and more different situations.

[2] complete is relative, due to the choice of description complexity. The more accurate we want to be, the more memory and processing power we need.

[3] a conceptual space is a multidimensional space, geometrical structure based on quality dimensions. *Quality dimensions correspond to the ways in which stimuli/features are judged to be similar or different. Judgments of similarity and difference typically generate an ordering relation of stimuli/features, e.g. judgments of level of control of the ball generate a natural ordering from $\check{S}weak\check{T}$ to $\check{S}trong\check{T}$.*

For a concrete implementation of case based reasoning[CBR], the chosen system is a description of the physical field (player, ball...) coupled with some abstract information such as the score, degree of possession of the ball, or even for future, a value describing an analysis of the opponents strategy.

To easily introduce belief revision in the RoboCup development, we started to implement possession belief that can be defined regarding the table below :

Possession	*degree*
no possession	0
no possession but in a scrum	1
possession but in a scrum	2
possession and clear	3
possession by the opponent team in the scrum	−2
possession by the opponent team and clear	−3

With this information, the dog should adopt a more defensive or aggressive tactic, regarding what the other teammate does. We now have to find some primitive facts that could be taken from sensors.

Here is our proposition :

Sentences	*Degree*
$\neg(possession \vee teammatehasball) \wedge clear \rightarrow deg\_poss\_-3$	1.0
$\neg(possession \vee teammatehasball) \wedge inscrum \rightarrow deg\_poss\_-2$	1.0
$\neg(possession \vee inscrum) \wedge teammatehasball \rightarrow deg\_poss\_0$	1.0
$\neg possession \wedge inscrum \wedge teammatehasball \rightarrow deg\_poss\_1$	1.0
$possession \wedge inscrum \rightarrow deg\_poss\_2$	1.0
$possession \wedge clear \rightarrow deg\_poss\_3$	1.0
$\neg seerobot \rightarrow clear$	1.0
$squeezeball \rightarrow possession$	1.0
$possession \wedge inscrum \rightarrow deg\_poss\_2$	1.0
$possession \wedge clear \rightarrow deg\_poss\_3$	1.0
$\neg moving \wedge walking \rightarrow inscrum$	0.8
$\neg seeball$	0.8
$moving$	0.8
$walking$	0.8
$\neg possession$	0.4
$\neg inscum$	0.3
$clear$	0.2

When primitives for current aibo are :

clear	$\triangleq$	has a free run to goal
inscrum	$\triangleq$	is surrounded by near opponent
moving	$\triangleq$	is actually moving
possession	$\triangleq$	possess the ball
seeball	$\triangleq$	can see the ball
squeezeball	$\triangleq$	can squeeze the ball
walking	$\triangleq$	is trying to walk

We can see something lacking in the definition of this language that we will discuss further in the needed improvements. For example, we need some possibilities of a valued argument to be unified. We can see it with the adjunction of constant predicate as deg_pos_i to cope this problem. Another problem we can quote is the "truth" entrenchment[1] (value 1.0) seems to be inefficient. The "always-true" rules are sometimes dismissed when an

[1] which is impossible in Saten system

inconsistency occurs. Usually, the system should block the new incoming information. This is the reason why we did not add some partition[1] rule for *deg_poss_i*.

We need to underline that the choice of a specific value for degree has been taken without testing it in real play. These values need to be optimized, following what kind of strategies we would like. These adjustments need to be done after a lot of testing. These values also depend on sensor accuracy. In Aibo we'll compute all the primitive variables in real-time. For example, the "clear" primitive depends on image quality, shape recognition, color detection ... With all these fixed variables, we can deduce a specific value for entrenchment. "SqueezeBall" in this sample is interpreted as an exception to the default value "possession".

Belief Revision is not very helpful in this example because a simple procedural language would be more efficient.

4.2.2 Psychology

Psychology can provide good examples to show the complexity of a belief revision server. This improved theory is especially interresting, due to it's similarity to the way we think, taking new information to grow experience. It is nearly the same process in an adjunction of a new fact in the theory base. Moreover, the given example can underline the importance of default values in such a theory. Here is a possible starting view of a soccer player's mind.

[1]by partition rule we mean a rule that prevents two different *deg_poss_i* to be true together.

Sentences	*Degree*
$Happy \rightarrow Out\_WagTail$	0.99
$Play \rightarrow Out\_StartSoccer$	0.95
$In\_BatteryJammed \rightarrow Sick$	0.9
$Sens\_BatteryFull \vee Sens\_Plugged \rightarrow Energy$	0.87
$\neg Sick$	0.8
$\neg In\_SadElse$	0.8
$Energy \wedge Stats\_Sporting \wedge \neg Sick \rightarrow Fit$	0.8
$WantPlay \wedge Fit \wedge \neg WorkToDo \rightarrow Play$	0.75
$Happy \rightarrow WantPlay$	0.6
$\begin{cases} (Stats\_Sporting \wedge Stats\_Winner \wedge \neg In\_SadElse) \\ \vee In\_HappyElse \rightarrow Happy \end{cases}$	0.5
$\neg Sens\_Plugged$	0.9
$\neg Sick$	0.8
$\neg In\_SadElse$	0.8
$Stats\_Winner$	0.8
$\neg In_{W}orkToDo$	0.5
$Stats\_Sporting$	0.5
$Sens\_BatteryFull$	0.3
$In\_HappyElse$	0.1

Where primitives[1] are :

$Sens\_BatteryFull$	$\triangleq$	's battery is full
$Sens\_Plugged$	$\triangleq$	is plugged on electricity
$In\_BatteryJammed$	$\triangleq$	's battery don't work
$In\_HappyElse$	$\triangleq$	is happy for another reason
$In\_WorkToDo$	$\triangleq$	has another job to do
$In\_SadElse$	$\triangleq$	is sad for another reason
$Stats\_Sporting$	$\triangleq$	does sport regularly
$Stats\_Winner$	$\triangleq$	has a good result ratio

We can then try to change some of these degrees, set to example values. These values were taken from our own belief of what a dog should think. These choices are then strictly instinctively set and personal. When we change any default or primitives to a larger value than any other rules using it as

[1] the *Sens* comes from sensors, the *In* comes from any other another process and *Stats* should come from a statistic method

an antecedent, we can see how the theory base evolves. In the case we want to have less reason to believe in a goal, this can also be done in decreasing the targeted value. Actually, the theorem prover will try to choose the least important action that could change for this result in all the decision's chain. Attributing a specific value in the beginning of the chain(on a primitive) is still a generate-and-test method for good behaviour. This method is good for optimizing values of entrenchment. Using produced value as statistics is also a fine choice and could be more effective with practise regarding the needed experience. This example is represented below by a graphical schema(See Figure 4.1).

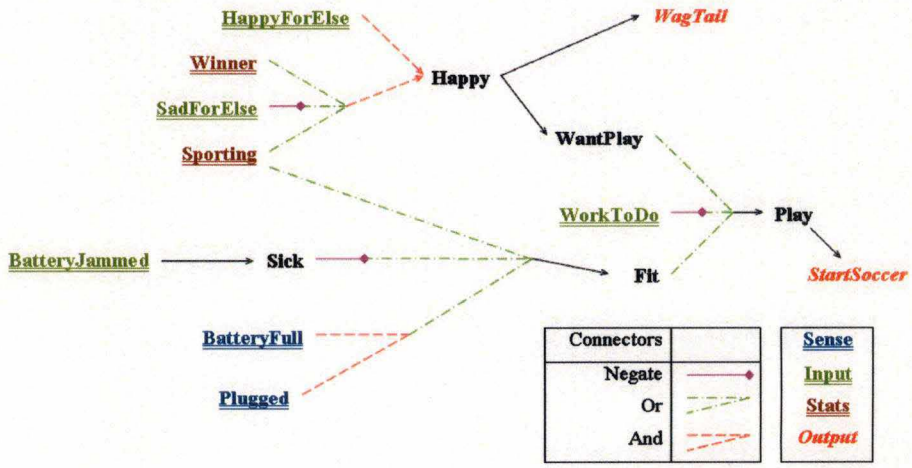

FIG. 4.1 – Psychology sample.

4.2.3 Bot

The third example we dreamed of was to code a bot, as in the entertainment games. Actually, it's a long and hard task and we just make a little part of this. The scenario we think about is a simplification of the CounterStrike game. It was in "save the hostage" game.
We used a bot[1] that didn't chase a SWAT followed by the hostage he was rescuing. Actually, the bot didn't understand because he did not see the SWAT when he was looking away, but he just saw hostages running. The rules from

[1]a bot is an artificial intelligence program with a human-like reasoning. It's the best to way to play a multiplayer when Internet is unavailable.

this game is for the good guy to save the hostages and the bad guy to prevent them from bringing them into freezone. The problem we underlined is that a hostage cannot move until a SWAT asks him to follow. By our simple human deduction, we know that a SWAT is here if a poor guy is running.
Let's describe an AI bot in belief revision to follow these simple rules :

1. Opponent team needs to rescue some guys.
2. Our team needs to "kill" the opponent.
3. There are x routes available to hostages.
4. The opponent's mate makes a lot of noise while running and a bit less when walking.
5. Obstacles[1] don't stop the noise(particular case).
6. Hostages can't move without a rescue guy.
7. The rescue mate can ask an hostage to follow him or stop.
8. Every guy knows the position of his mate by a radar system.

We tried to implement a logical system for a bad guy bot. It is just an example in first-order logic. Only Saten prover can presently manipulate these kind of information. In fact, we can in particular case implement it in propositional sentences.
We had some trouble using Saten, but in further development this problems should be corrected. Here is our example[2] :

Sentences	*Degree*
$\forall x.seeswat(x) \rightarrow swat(x)$	0.90
$\forall x.seefoe(x) \rightarrow foe(x)$	0.90
$\forall x.seehostage(x) \rightarrow hostage(x)$	0.90
$\forall x.radar(x) \rightarrow swat(x)$	0.90
$\forall x.hostage(x) \wedge \neg foe(x) \wedge \neg swat(x) \rightarrow \neg move(x)$	0.90
$\forall x.noise(x) \rightarrow hostage(x) \vee foe(x) \vee swat(x)$	0.90
$\forall x.foe(x) \rightarrow attack(x)$	0.90

Where primitives for vicinity of position x and current terrorist player are

[1]Walls in general, avoiding to look and walk through
[2]Note that in this sample, x is a given place

$seeswat$	$\triangleq$	sees that a SWAT is present
$seefoe$	$\triangleq$	sees that a mate is present
$seehostage$	$\triangleq$	sees that a hostage is present
$radar$	$\triangleq$	radar is blinking there blinking
$swat, foe, hostage$	$\triangleq$	resp. SWAT, foe, hostage is present
$noise$	$\triangleq$	belief[1] of a sound coming from there

This example has not been really implemented due to lack of stability in Saten for the first-order logic. As we continue to develop a belief revision service, these improvements should soon correct it. Here, the default values are not already chosen but are intrinsically present. We can deduce by our own reasoning that this knowledge base coupled with another script system for other intelligence skills is a good step to an autonomous intelligent system.

This method has a major feature in the easy and automatic introduction of rules corresponding to the established rules for a game. We describe all rules in natural language then converting them to a strong belief in our knowledge base. We can easily determine which facts or rules are more "believable". For instance, sound is less important than vision. At least, these are our current thoughts. It is for others to judge which sentences to use in constructing a theory base. A set of academic samples could be created to this end.

However, some theories we elaborate don't give the expected result.The hardest thing to determine is using wether or not we are working with a good description. A lot of studies are in progress in the area of nonmonotonic reasoning. We also think that practise is strongly recommended.

[1] varying also with volume level

4.3 Future Development

There are many ways to use belief revision. We tried to determine two of them that could be useful to develop.

Search Engine

All internet search engines use a ranking between all spidered websites. They link the given keywords to pages with a degree of correspondence. This feature could be generated by a belief revision server. The interest is not only in the simple coding and revision of rankings but also in developing a logical network. This method could lead to more logical choices for sites and can lead to a decrease in the space needed for link information.

Breakdown Management

We could implement a system that would build up a statistical model of the symptoms that leads to failure. This is a long-term process which could lead to an expert system without any human decision.

Chapitre 5

Conclusions

5.1 Needed & Recommended Improvement

5.1.1 Belief Revision Improvement

This part concerns the improvement that can be done by the belief revision system provider.

First-Order Logic Improvement

It is essential that first-order logic works properly on any belief revision server. The problem is that only Saten implements it and we had a lot of trouble with it. AIFS didn't implement it at all.

Remember our example about case based reasoning in the previous chapter, we used some constant predicate to define explicitly and separately the different value of a degree of possession as $deg\_poss\_1, deg\_poss\_2, ...$

In first-order logic, we could use a more general predicate as $degree\_possession(x)$ where x is the value of possession. In such case, the system has to instantiate every possibility of x and check the corresponding one . We call this operation the unification of a variable x.

Several programs are now able to resolve this simple problem like the well-known Prolog[1] or like the CSP[2]-provers, Eclipse[3] for instance which can unify x with all possible values. Then we can select the best solution from this set. The principle seems simple but it is a time consuming task.

Withdrawal vs Contraction

The systems we use have in their public interfaces what the theory base calls *revision*. To "erase" something from the dog memory, we have to revise the sentence to a degree equal to zero. That implies a contraction of the rule to be executed. If you remember of the former belief revision explanation, a contraction change not only the sentence's belief but also all the related part of this theory base. The possibility to remove a belief without changing any other sentences could useful, in particular when we are learning belief revision applications. In other words, withdrawal is necessary.

This could be easily done in AIFS system, by using integrated method. Saten could be more complicated because even the simple revision to the $zero^{th}$ degree is impossible due to the restriction of (0,1) entrenchment.
Here comes the reason we didn't change it ourself. The documentation is not sufficient and we don't know any side effect of modifying such an internal method.

5.1.2 Linked Features

This part concerns all plug-ins that could be linked with the belief revision system. It can be done by either side : the belief revision system provider or intelligence server.

Monotonic Reasoning

We need to use a part of "always-true" sentences in several examples. In the current system known as AIFS, we can use some entrenchment of 1.0 that should correspond to an indefeasible rule. Sadly, this entrenchment is equivalent to any other entrenchment and thus, can be defeated. The required property for an always true sentence is to be non deferrable. In theory, if an inconsistence occurs, we can't withdraw such a sentence. We have to

[1] logic programming, several version exists
[2] Constraint Satisfaction Problems
[3] http ://www.icparc.ic.ac.uk/eclipse/

block the action and return a feedback to the requesting user.

This feature is the major fact of the monotonic logic that we removed in our development, as we wanted a dynamic theory. After practising with some examples, our viewpoint is that in a global application we need both reasonings at the same time. The reason is that we need to fix some global rules to guide the reasoning where we want it to go.
Here is a partition example to quickly illustrate it [1] :

$$a_i \; xor \; a_j \text{ when } i \neq j$$
$$s_i \rightarrow a_i \quad \forall i$$

In this example, we don't want the system to choose more than one i such that a_i is true[2].

Calculus and Arithmetic Manipulation

As we need to compute information about existing external facts, it could be useful to implement some basic calculus with the possibility of assigning other types of variables.

Without any additions into value domains, we could accept comparing some belief, to each other, that could be useful in generating a more comprehensible way to characterize a situation. We could integrate some special reasoning with semi-linked concepts. Here is an example : "IF I prefer to play football when the sun is shining than playing video games when the weather is cold, THEN I prefer doing sport than playing inside games". This reasoning could be some weird but the main psychological tests in Internet communities[3] group are done with such a method.

The best issue is to permits use of some simple variable type as integer, real, string ... to be linked by any variable taken from external program. These variables could then be compared inside this program and used in reasoning. Here is an example that we could have used in Case Based Reasoning :

$$(match\_won > match\_lose) \rightarrow Stats\_winner.$$

This kind of development should be made at the intelligence server to keep the maximum amount of computation in an external processor, to avoid stressing Aibo's CPU.

[1] note : $a \; xor \; b \equiv (a \vee b) \wedge \neg(a \wedge b)$
[2] as in the example with deg_poss_i
[3] like www.match.com, www.msn.com...

Sentence Description

For any users working with pre-existing theory database, we recommend assigning a written description of what each sentence gives to the knowledge base.

5.1.3 Software Improvement

This part concerns the improvements need to be done to our server.

System Translation

Some remaining problems persist in the conversion of the undocumented syntax used by both systems. A good way of implementing it is to convert them into a more abstract syntax. Anyway, we recommend creating a precise definition of all formalisms for making all conversion.

Knowledge Readability

Sentences may be hard to read and understand in the actual system. The first thing to do is separate facts and rules. Splitting antecedent and consequent should be a better idea. To this end, we need to think about another way to represent any sentences because some sentences have more than one implication. We then need a dynamic representation as trees or something else to represent information in each level.

Client's Server Crash Management

When the server crashes for any reasons, we can suffer some problems within the request's processing. This problem should be sorted out in next release of this program.

Watcher Add-on

We could also add some useful watch options as in most GUI programming interfaces. The main idea is to define some sentences that we don't need to revise, but we want to keep an eye on their degrees. This is really more readable and doesn't use any space in the theory base.

Knowledge Subset

The online version of Saten implements a good method to understand and play with knowledge base more easily. When you want to revise a theory, you

can just select some of the axioms of the theory and revise it as it was just an independent subset of the theory base. This could be added in further development.

5.2 Reached Results

5.2.1 Belief Revision

Belief revision seems a good idea for implementing some human-like reasoning that could be involving, changing and very complex. This system is able to construct step by step a complex behaviour with a certain structure. We can help ourselves by defining a large knowledge base, using a system based on arc consistency or other techniques to represent knowledge networks before implementing them. We've seen such applications in this report **??**. There are still improvements to be done in selecting the best adjustment for a given situation

With our developments and practise, we conclude that we need both monotonic and nonmonotonic reasoning to create a knowledge base and to keep control over it. Several rules need to be set once for all. The way we are working in our mind is still a nonmonotonic reasoning but we are also able to make some mistakes in interpretation. This is the greatest difference between our reasoning and the nonmonotonic reasoning. Any rules that we can accept in our mind that are not sure and may be false for reasons other than just logic, for example interpretation. A computer will not make the same mistake as we define rules totally explicitly, with mathematical formalism. Such description are not fuzzy. Anyway, we are agreed with the need of rules we can revise. For instance, we can set some physical rules, "earth is flat" or "weight is 9.81 * mass by default", and then revise it for all facts or for a subset of the targeted domain("earth is a sphere" and "9.81 is only for earth").

5.2.2 Server

The use of an intelligence server seems to be a good idea. It is a good solution to all problems with such volatile informatics systems. Several people tried to break the problems we encountered and didn't succeed. Then this solution is clearly one of the best solution until we had enough tools provided by Sony to play with.
The server is still in improvement as we continue its development and it will

continue unless the project is not used. There are still some problems but the main features works perfectly.

5.3 New Possibilities

The difficulty is in taking these theories into applications worths the case. We really believe in the future of it, even if we just approach the problem in the surface. The different possibilities to model the human brain are very exciting and promising.

Bibliographie

[DPO94] D. Poole, *Default Logic* - in Handbook of logic in artificial intelligence and logic programming (vol. 3) : nonmonotonic reasoning and uncertain reasoning,Oxford University Press 1994

[HAY73] P. Hayes, *The Frame Problem and Related Problems in AI. Artificial and Human Thinking*, A. Elithorn and D. Jones (eds.), Jossey-Bass, 1973

[LIF94] V. Lifschitz, *Circumsciption* - in Handbook of logic in artificial intelligence and logic programming (vol. 3) : nonmonotonic reasoning and uncertain reasoning,Oxford University Press 1994

[MAK94] D. Makinson, *General Patterns in Nonmonotonic Reasoning* - in Handbook of logic in artificial intelligence and logic programming (vol. 3) : nonmonotonic reasoning and uncertain reasoning,Oxford University Press 1994

[WEI76] J. Weizenbaum, *Computer power and human reason : from judgment to calculation.*, Freeman, 1976

[AGM85] Alchourron, Gardenfors and Makinson, *On the Logic of Theory Change : Partial Meet Contraction Functions and Their Associated Revision Functions*, Journal of Symbolic Logic 50(510–530), 1985

[JMJ-C] TIA, J-M Jacquet, 2002-2003

[PYS] NonMonotonic Reasoning, P-Y Schobbens, 2003-2004

[SADJ] Williams and Sims, SATEN, *An object-oriented web-based revision and extraction engine*, The University of Newcastle(Australia), 2000

[AIFS] Lau, ter Hofstede and Bruza,*A Study of Belief Revision in the Context of Adaptive Information Filtering*,QUT university(Australia),1999

[CBR] [CBR] Karol, Nebel, Stanton, and Williams*Case Based Game Play in the RoboCup Four-Legged League Part I The Theoretical Model*, UTS(Australia),2004

[UUR02] Chalup, Creek, Freeston, Lovell, Marshall, Middleton, Murch, Quinlan, Shanks, Stanton, and Williams, *When NUbots Attack ! The 2002 NUbots Team Report*, The University of Newcastle(Australia), 2003

[UUR03] Agnew, Brownlow, Dissanayake, Hartanto, Heinitz, Karol, Stanton, Trieu, Williams and Zeman, *Robot World Cup Soccer : The Power of UTS Unleashed !*, UTS(Australia), 2004

[WIL95] M-A Williams,*Applications of Belief Revision*, The University of Newcastle(Australia), 1995

[DirectIA] http ://www.directia.com/

[Flux] http ://www.fluxagent.org/

[GCJ] http ://gcc.gnu.org/java/

[JNI] http ://java.sun.com/j2se/1.3/docs/guide/jni/

[MASA] http ://www.masagroup.net/

[MAT-W] http ://mathworld.wolfram.com/

[MYWWW] http ://www.info.fundp.ac.be/ỹvanders/m̀emoire/Samus_Believer/

[OPENR] http ://www.openr.org/

[SATEN] http ://magic.it.uts.edu.au/systems/saten.html

[SemWeb] http ://www.semanticweb.org/

[TUR36] http ://www.turing.org.uk/

[W3CSW] http ://www.w3.org/2001/sw/